3

AFIT/GCS/ENG/93-09

A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions

THESIS

Michael Thurman Gardner
Major, USAF

AFIT/GCS/ENG/93-09

21808

AFIT/GCS/ENG/93-09

A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Systems)

Michael Thurman Gardner, B.S.

Major, USAF

December, 1993

Acknowledgements

If I have seen farther than others, it is because I have stood upon the shoulders of giants. Sir Isaac Newton

This thesis would have not been possible without the advice and support of others. One of the giants in this effort has been my advisor, Lieutenant Colonel Phil Amburn. I am deeply indebted to him for his foresight and enthusiasm. Another giant, Lieutenant Colonel Dave Neyland of ARPA/ASTO, provided me with the opportunity, as well as the funding, to pursue one of my dreams–research that could provide a significant impact on the operational Air Force. Of course this effort could not have been accomplished without the full support and cooperation of Skip Buss, Mike Mateyka, and the commanders of the 57 Wing, Nellis Air Force Base.

Special thanks to Bruce Clay, now working at SAIC, who developed the hardware for the computer communications components of this thesis project and was responsible for all of the network interface software. He also was the major contributor for the data translation software. I could not have completed this project without his assistance.

There are many others who made significant contributions to this effort. My sincere appreciation goes to Steve Sheasby for his *Object Manager*, Mark Snyder for the *ObjectSim* framework, Dean McCarty for his digital terrain format conversion software, Bill Gerhard for the generic head-up-display used in the cockpit views, Matt Erichsen for the round earth utilities which made the transition from "flat earth" to "round earth" easier than a voyage across the Atlantic. I'm also grateful to the "night shift" members–Brian Soltz, Kirk Wilson, and Andrea Kunz. They were my sounding board and comic relief during those early mornings in the lab.

Additional thanks go to Charlie Powers for his "Klingler-like" ability to procure the necessary hardware when I needed it. I am also thankful to Gerald Laetz and Bill Saucier of Computrac Corporation for their expert advice on the RFMDS system.

I also appreciate the time and effort that Lieutenant Colonels Martin Stytz and Patricia Lawlis devoted to reviewing this thesis. Their support of distributed simulation research at AFIT has created a stimulating environment for creativity.

Last, but not least, this entire experience at AFIT would not have been possible without the love and support of my family. I cannot thank my wife, Pat, enough for her expertise in editing this work and providing the gentle nudges to keep going when the going got tough. My children, Carolyn, Robert, and Heidi reminded me that life outside the AFIT engineering building is full of adventure. They always included me in their prayers and kept me focused on the Source of my blessings.

Michael Thurman Gardner

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCS/ENG/93-09

Abstract

Air Force leaders, recognizing the need for improved training following the Vietnam War, implemented the Red Flag exercises at Nellis AFB. At the heart of this training is the Red Flag Measurement and Debriefing System (RFMDS) and its capability to accurately reconstruct the elements of an intense exercise fought over the deserts of Nevada. This thesis uses the technology of distributed interactive simulation (DIS) to transmit aircraft telemetry onto computer networks, allowing the monitoring and analysis of live Red Flag missions at any site with compatible communications equipment and thesis software. The use of standard DIS protocols enables simulators to "see" Red Flag operations. The three components of the Remote Debriefing Tool bridge reality with simulation. The computer communications component serves as a transparent tap into the RFMDS data stream. The data translation component translates this information into DIS protocol data units (PDUs). PDUs received by the interface and visualization component are used to generate two- and three- dimensional images of the Red Flag environment onto a Silicon Graphics workstation. An extensive set of analysis tools combined with a graphical user interface allows reconstruction of airborne activities, producing more effective and comprehensive training.

A Distributed Interactive Simulation Based Remote Debriefing Tool for Red Flag Missions

I. Introduction

1.1 Overview

Twenty-five years ago, in early 1968, the United States was heavily involved in the Vietnam War. Unlike previous wars however, the air-to-air exchange ratio of our pilots and aircraft had dropped significantly. During the World War II years of 1942-1945, air operations against enemy aircraft achieved an overall exchange ratio of 14:1. In Korea, Naval aviators managed a 3.2:1 kill-loss ratio despite having to fight enemy jet aircraft with piston-driven fighters. But in Vietnam, the overall Navy exchange ratio had dropped to 2.3:1. Statistics from the other services were no better. Combat kill ratios were the worst in the history of U.S. aerial warfare (2:36).

Captain Frank W. Ault, USN, was given the mandate to find out why the Navy was not shooting down more MiGs and to recommend what should be done about it. He created five expert study teams to examine the Navy's fighter weapon systems from "womb to tomb." This examination of the Navy's weapon system life cycle and the 242 recommendations that arose from Capt Ault's study are contained in a report that was delivered to the Naval leadership in 1969 (2:37).

Among other findings, the report indicated that it was not uncommon to find a combat situation in which neither the pilot nor the aircraft he was flying had ever launched a live missile. In addition, the report cited a change in combat philosophy that had shifted to placing more reliance on the machine than on the man. As a result aircrews were required to "conduct a heads-up fight with a heads-down system" during man-to-man aerial combat (2:38). The implications were that training and combat readiness were real problems.

These findings, among others, suggested the need for improved combat readiness training and an air combat maneuvering range (ACMR). Thus the findings of the Ault Report were the genesis for today's Navy Fighter Weapons School (*Topgun*) (2:38).

Research conducted along parallel lines by the Air Force following the Vietnam war, yielded some of the same conclusions–aircrews received inadequate training for combat. Air Force actions to remedy these deficiencies included the creation of the Aggressors in 1972. This organization was an unprecedented opposing "Red Force" for our fighter and attack aircrews to train against during peacetime. While the Aggressors provided a unique capability to train against an active opposing force it did not replicate the entire Soviet air and ground threat (16:33).

Red Flag was conceived in 1975 to provide realistic simulated combat missions for aircrews to support three main goals (16:33):

1. train like we plan to fight;
2. train with our allies and sister services;
3. train to develop the readiness to deter war and win any conflict.

To accomplish these goals the ranges north of Nellis AFB were converted into a simulated Soviet satellite country, complete with anti-aircraft-artillery (AAA) and surface-to-air missiles (SAMs) protecting both tactical and strategic targets. A sophisticated $58 million aircraft tracking system was installed to track all aircraft during the mock combat. This system is known as the Red Flag Measurement and Debriefing System (RFMDS) (16:33). (The ACMR concept recommended in the Ault Report and developed by the Navy was the precursor not only to the Air Force's air combat maneuvering instrumentation (ACMI) range, but also to the RFMDS.)

At Red Flag, aircrews plan and then execute simulated combat missions into a dense threat environment. They are required to defend themselves and their flight members from both air and ground threats, acquire and successfully attack their assigned targets, and recover safely. Each mission exposes the aircrews to stressful, combat-like scenarios and develops the skills necessary to survive in actual combat. Historically, most losses occur during a pilot's first ten combat missions (16:33) thus completion of the Red Flag experience is designed to be roughly equivalent to the level of experience gained during a pilot's first few weeks of combat and pilots enter combat at higher levels of readiness.

The debrief which follows the mission is almost as important as the mission itself. During this evaluation all participants have an opportunity to critically examine every aspect of their flight performance and to learn how to improve their flying skills and judgement for future missions. At the heart of the debrief is RFMDS. This tool is used to graphically reconstruct the mission by displaying aircraft flight paths, rebroadcasting radio communications, and replaying video tapes from "enemy" ground based threats which depict the effectiveness of aircrew evasive maneuvers.

Not all aircrews in the Red Flag exercises takeoff from and return to Nellis AFB. Those aircrews landing at other bases do not have the opportunity to participate in the mass debriefs and to use the state-of-the-art technology of the RFMDS. As a result, mission activities that take place outside of the remote aircrew's field of view go unnoticed. This places an unnecessary handicap on these aircrews as they try to reconstruct the mission with limited information. In order to provide instructive debriefs at the remote sites, aircrews need to "see" the same things that are presented at the mass debrief.

According to Mr. Mike Mateyka of the 414th Test Squadron at Nellis AFB, the Air Force recognized the need and, in 1989, established a requirement for a remote debriefing system that mirrored the capabilities of the RFMDS. A prototype demonstration version developed by Cubic Corporation, called the Remote Debriefing Display System (RDDS), satisfying the Air Force requirement was used in two tests in 1990. The first test was conducted at Nellis AFB and the second at a remote site in Indian Springs, Nevada. Pilot reaction to the RDDS was favorable and greatly improved aircrew abilities to reconstruct aerial engagements. It provided a level of situational awareness that was previously unavailable at remote sites. This prototype was the forerunner of the Mini DDS that is currently under development and scheduled for delivery in 1994. Limited funding will allow the acquisition of only three units.

In addition to Red Flag's needs, the 57th Test Group at Nellis AFB has a desire for a flight test analysis tool that can be used for their test missions in their own facilities. (This analysis must currently be done at the Red Flag building.) Some of the test missions involve aircraft and crews from the 99th Bomb Wing at Ellsworth AFB, South Dakota. At present, personnel from the 99 BW deploy to Nellis and monitor test missions with the RFMDS. Considerable time and money could be saved if the capability existed to monitor 99 BW aircraft missions over the Nellis ranges from Ellsworth. Research conducted in the area of distributed interactive simulation (DIS) provides a means to satisfy the communications requirements of the needed remote debriefing tool.

The Advanced Research Projects Agency (ARPA) has sponsored simulation research within the defense community for over a decade (10:1). According to Lt Col Dave Neyland, ARPA/ASTO, this research has traditionally focused on two major areas: (a) force-level simulations, or wargames, which test decision makers' abilities to judiciously commit forces without large expenditures of men and material (12:44); and (b) distributed interactive simulation which allows multiple operator or semi-autonomous-force based simulators connected over a wide area network to interact with each other.

The ability of simulators to interact with operational weapon systems forms a newer, third area of research. ARPA-sponsored efforts in this area have already demonstrated a limited capacity for operational vehicles and simulated entities to interact within the same virtual environment. During a SIMNET orientation briefing at Fort Knox, Kentucky, the Navy's Battle Fleet In-Port Trainer was described. In this exercise, pilots flying helicopter simulators exhibited the ability to take off from a simulated base, fly to a simulated Naval port and land on one of the ships. At the same time, the real Navy ship participating in this exercise was able to acquire, track and display the helicopter simulators on its real radar equipment. These demonstrations represent only the beginning of the research effort necessary to bring simulation and operational vehicles together in a common environment. Simulator interactions with operational aircraft flying on live ranges is another facet of this new area of research.

Together, these three research areas: wargames, distributed interactive simulation, and live range interaction, form what has been called the ARPA simulation triangle, and

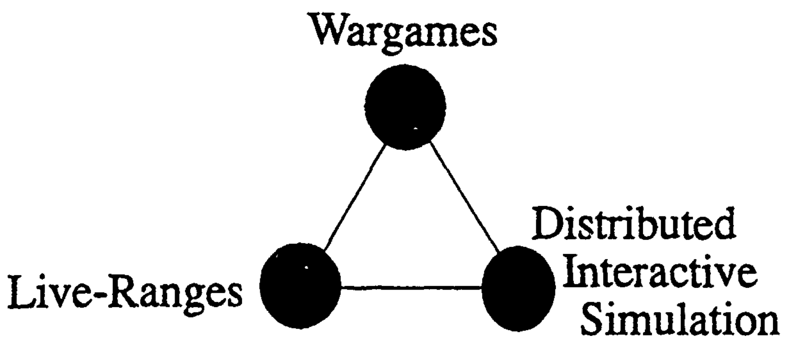

Figure 1. ARPA Areas of Simulation Research

represent an overall vision of simulation research for the future. (See Figure 1) Significant progress in all three areas may one day provide a means for thousands of simulators, ships, tanks, infantry, and aircraft to participate together in "truly" large scale exercises.

Dr. Earl A. Alluisi, Office of the Director of Defense Research and Engineering, echoed a similar vision when he stated,

> We have a vision. The vision is of an interactive network that can be used for training individual troops, sailors, marines, and airmen, as parts of the crews, groups, teams, and units to which they belong. The network is the key...
>
> Thus, the Army's tank crew may have a relatively low-cost graphics-based simulator... The flying squadrons may have more costly and more complex computer-image-generated displays in their simulators. But with a proper net, both types can be hung onto the net, and can be used to train combat tactical skills in simulation of many-on-many situations. With appropriate distribution of such capabilities, the interactive network can be used in the training not only of individuals operating vehicle battle stations, but also of individuals in command-and-control battle stations – the platoon or flight or individual ship, at one end, and with the expansion to wargaming capabilities, the theater commander, his staff, and his component commanders and their staffs, at the other end(1:3).

According to Lt Col Dave Neyland[-1z, ARPA/ASTO, "This thesis project represents one of ARPA's first efforts to explore the issues involved with integrating live aircraft into Distributed Interactive Simulation by broadcasting aircraft telemetry data over computer communication networks in a format that can be used by other compatible ground-based simulators." These simulators would then have an ability to monitor aircraft activities on a live range and move around in an environment where both computer-generated and real-world entities exist. Additionally, this project provides a debriefing tool suitable for remote location analysis of air-to-air missions flown on the Nellis ranges and an opportunity for the 57 TG to remotely monitor its own test missions.

1 2 Thesis Statement

A hardware and software system can be developed which proves the feasibility of implementing a Red Flag mission monitoring and debriefing tool that utilizes the DIS communication protocol for transmitting aircraft telemetry to remote sites and provides an interactive interface to a state-of-the-art three-dimensional image generator. (This system will subsequently be referred to as the remote debriefing tool (*RDT*)). The *RDT* can be constructed with off-the-shelf network communications hardware and thesis software and is therefore dependent only upon the availability of a suitable workstation and communications equipment. The conceptual configuration of the following requirements is depicted in Figure 2. To establish the feasiblity of such a system, the *RDT* must demonstrate the capability to do the following:

1. Monitor RFMDS communications and extract those messages which contain aircraft telemetry (aircraft position, flight parameters, and weapons control status);
2. Translate the RFMDS telemetry into DIS message formats, known as protocol data units (PDUs);
3. Transmit the DIS PDUs onto a computer communications network;
4. Read DIS PDUs from the network and translate the information into a form usable by the computer image generation software;
5. Display two- and three-dimensional views of aircraft flying on the range in a format suitable for mission monitoring and debriefing;
6. Provide a simple and intuitive interface to control the various views;
7. Archive the data onto non-volatile storage; and

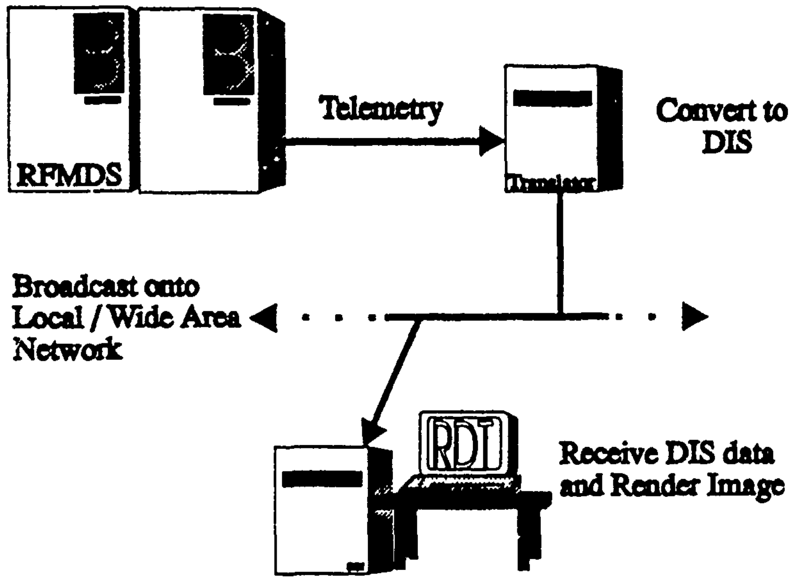
RFMDS
Telemetry
Translator
Convert to DIS
Broadcast onto Local / Wide Area Network
RDT
Receive DIS data and Render Image

Figure 2. RDT Conceptual Configuration

8. Retrieve archived data and retransmit it over the network for a mission replay capability.

1.3 Scope

The scope of this thesis will be limited to the research, software design and implementation necessary to provide a basic debriefing system that can be used at remote locations. (By definition, a basic debriefing system will mean a system that is only suitable for monitoring and replaying air-to-air and surface-to-air mission elements.)

Central to the development of the *RDT* is the need to communicate over computer networks using the DIS protocol. (This protocol is described in the following chapter.) The DIS standard defines numerous PDU types which provide for the exchange of entity information as well as simulation exercise management(23:22). The scope of this project limits the implementation of PDU types to those required for aircraft entity interaction. These PDU types include the entity state, fire and detonation PDUs. Simulation management PDUs fall outside the purview of this project and are not implemented.

The computer generated displays of the *RDT* mirror the major capabilities of the RFMDS but do not fully emulate all its features. Additionally, radio communications present in the RFMDS are not incorporated into the initial version of the *RDT*. Project time constraints did not allow implementation of audio, although the DIS standard provides a means for radio communications between entities.

1.4 Assumptions

This section enumerates the assumptions that were made in developing the thesis statement and scope of this project.

1. The principle user of the *RDT* is someone interested in an accurate range monitoring system that is capable of tracking and recording aircraft movements but who does not require a level of detail necessary for a full engineering analysis. Aircrews participating in the Red Flag exercises are the prime target for this project.
2. The computer image generation hardware for the *RDT* is a Silicon Graphics IRIS 4D workstation. This meets the "low-cost" definition as expressed by sponsors at Nellis AFB. Multiprocessor capabilities improve image generation speeds without altering the basic functionality of the *RDT* software.
3. Existing computer network communication devices are able to transmit and receive Ethernet packets and DIS PDUs.
4. The use of AT&T C++ or C languages is compatible with existing software applications.
5. The Silicon Graphics machines use a UNIX operating system derivative–IRIX 4.0.5 or later.
6. The third draft of the DIS version 2.0 proposed IEEE standard is the guide for generating and interpreting DIS PDUs. (This draft is accepted by the DIS community as the standard for applications in 1993.)

1.5 General Approach

Logically, the *RDT* can be divided into four major areas of effort: computer communications, data translation, user interface design and computer image generation. Within each of these areas, a significant number of tasks must be completed before the entire system can function as intended. Each of the major areas, and the tasks they encompass, is described in the next few sections.

1.5.1 Computer Communications. Computer communication forms the key area of the entire project. The premise of broadcasting live telemetry onto a distributed simulation network cannot be accomplished without the ability to electronically capture the data from the RFMDS and then retransmit that data over local and/or wide area networks. The tasks necessary to establish the proper communication links are listed below.

1. Install a transparent connection onto the RFMDS that will allow data monitoring without any message acknowledgements. This ensures that the *RDT* cannot interfere with any RFMDS operations or be responsible for any system difficulties.
2. Monitor the RFMDS data stream and capture the data messages.
3. Convert the data messages into Ethernet packets for retransmission onto a local network.

1.5.2 Data Translation. Data translation involves mapping the RFMDS data messages into the proposed DIS standard PDU formats. In order to accomplish this function the following tasks must be completed.

1. Determine the specific RFMDS message types and formats tha. will provide the level of information necessary for the image generation.
2. Define a mapping that will be used by the translation software to convert the RFMDS messages into DIS PDUs.
3. Filter out unneeded messages from the RFMDS data stream and translate the required messages into DIS PDUs. (This translation software will need to interface

with network communications software in order to broadcast the DIS PDUs onto a distributed simulation network.)

1.5.3 User Interface Design. Because the *RDT* computer image generation is implemented on a single workstation, a significant effort was expended in order to capture and mirror the complex functionality of the multiple RFMDS displays and controls. The tasks necessary to develop a suitable user interface are:

1. Determine and then use a suitable interface design methodology that can be applied to the development of the RDT.
2. Select from the many available input devices, such as mice, keyboard, spaceballs, joysticks, and data gloves, those which provide the needed functionality for a set of simple interface controls.
3. Find a set of graphical user interface tools that can be used to quickly develop a user interface prototype. An interface prototype is a rapidly created version of some or all of the final interface, generally with limited functionality (5:430).
4. Through a series of user tests and design modifications, develop the final user interface configuration.

1.5.4 Computer Image Generation. Once the DIS-formatted mission data is available at the workstation, a computer image can be generated which depicts the operations of aircraft flying on the Nellis ranges. The following tasks are necessary to develop these images.

1. Determine the required set of display capabilities that are needed for a usable mission debriefing tool.
2. Evaluate available computer image generation packages to determine the feasibility of modifying them for use as the primary image generation software.
3. Develop an image generation software toolkit that can be used to provide the needed views for the RDT.

4. Integrate the image generation software with the user interface so that the views may be controlled as desired. Iterate through a series of user tests and design modifications until a suitable *RDT* final configuration is achieved.

1.6 Additional Thesis Support

The philosophy of this thesis project is not to develop as much new software as possible, but to use existing software where it is available and compatible and then add to its functionality where required. In keeping with this overall development concept, this section introduces software available at the Air Force Institute of Technology (AFIT) that provides many of the needed capabilities described previously.

1.6.1 Communication and Data Translation. Communications software written by Bruce Clay exists at AFIT in the form of a set of network daemons. These daemons are system processes which remain dormant until a request to transfer data over the communications ports is received. One of the daemons is responsible for transmitting data onto the network while the other daemon receives data from the network and makes it available to application programs. This software was originally created for use with SIMNET PDUs and was subsequently modified to broadcast and receive DIS PDUs. In addition, Bruce Clay compiled a set of routines which were used to generate and monitor Ethernet communications.

1.6.2 Object Manager. Sheasby (18) developed a set of software methods to manage all of the information being broadcast over the network about individual entities participating in a distributed simulation exercise. This collection of C++ classes, called the *Object Manager* inserts, updates, traverses, and deletes entities from an entity class structure hierarchy. Additionally, the *Object Manager* performs the dead reckoning calculations necessary for any moving vehicles (18:3). Because the *Object Manager* was originally developed for use with SIMNET, modifications were made to make it compatible with the new DIS standard.

1.6.3 ObjectSim. Snyder (20) describes the construction of an application framework which can be used to render models within a visual simulation environment. This framework grew from the desire to reuse concepts and ideas within the graphics lab at AFIT from year to year without having to "reinvent the wheel" for each new application. *ObjectSim* provides a set of high level functions within its object-oriented class structures that frees developers from the details of rendering geometry on the Silicon Graphics machines and allows them greater time to deal with other simulation intricacies. This thesis project and five additional thesis efforts, (4), (6), (21), (29), (9), used the *ObjectSim* framework during the 1993 academic cycle at AFIT.

The *Virtual cockpit* (VC) is a flight simulator designed for use in an interactive multiplayer environment. The cockpit is constructed with commercial off-the-shelf equipment including a head-mounted display, a hands-on throttle and stick, and a Silicon Graphics workstation for the image generator. Sensor subsystems developed by Erichsen (4) display the location and orientation of network players. Air and ground weapon systems brought on-line by Gerhard (6) allow the *Virtual Cockpit* to employ bombs, guns and missiles in the simulated environment.

The *Synthetic Battle Bridge* (SBB) (29), (21) presents an immersive, simulated environment that is used to observe distributed simulations from a battlefield commander's point of view. The SBB is capable of interfacing to a variety of head-mounted display devices and a Fake Space System's BOOM.

The *Satellite Modeler*, using the *ObjectSim* framework, was developed by Kunz (9). This application depicts satellites in orbit around the earth and allow users to interact with the satellite models. The *Satellite Modeler* provides an opportunity to explore configurations, orbital elements, and procedures which were previously very difficult to visualize.

Because each of these applications was built around a common framework, ideas and methods were shared between them which provided additional opportunities for bringing features and functionality to the RDT that might otherwise have been impractical because of time constraints.

1.7 Thesis Presentation

This thesis is divided into seven chapters and appendices. Chapter II presents background information relevant to this thesis project. In particular, the chapter provides insight into the concepts of distributed interactive simulation, RFMDS, Silicon Graphics image generation software and computer communications. Chapter III discusses the design of the computer communications and data translation areas while Chapter IV describes the design of the user interface and image generation software. Chapter V details the implementation of the complete *RDT* software system. Chapter VI discusses the performance characteristics of the software. Chapter VII recaps the thesis project and proposes recommendations for future work.

II. Background

This chapter lays the foundation for understanding the concepts of distributed interactive simulation that are used within this thesis project. In addition, a description of the RFMDS provides an overview and the technical details necessary to understand how the *RDT* interface to RFMDS can be achieved. The final section of this chapter outlines the characteristics of a software rendering library developed by Silicon Graphics, Inc. This library, known as *Performer*, comprises the basic toolkit for generating the 3D computer images necessary for a visual simulation application such as the RDT.

2.1 Distributed Interactive Simulation

2.1.1 Evolution. Distributed interactive simulation had its beginnings in 1983 when the Defense Advanced Research Projects Agency (DARPA now called ARPA) initiated a program to enhance tactical team performance via distributed simulator networking (SIMNET). The first conceptual demonstration was conducted in 1984 following completion of the initial system design (12:1). Thorpe reports that SIMNET was used to train U.S. troops for the Canadian Army Trophy (CAT) Competition in 1987. (CAT 87 was the first year that a U.S. tank team placed first.) The use of SIMNET to familiarize participating units with the exercise range and improve crew coordination, team interaction, and command and control skills appeared to enhance the performance of U.S. Army tank units (27:266). Subsequent tests performed in 1987 at Grafenwoehr, Germany and Fort Benning, Georgia confirmed that the SIMNET technology "holds potential for both readiness and new system development and acquisition (27:272)." By the end of 1987 helicopter simulators had been delivered and installed at Ft. Rucker, Alabama and tank simulators were likewise in place at Ft. Knox, Kentucky. At the completion of the program there were a total of 250 simulators installed at more than 9 locations (12:1). SIMNET technology is still used today to train U.S. Army tank teams around the country.

SIMNET was intended to provide a method for simulating battles involving many vehicles by interconnecting large numbers of interactive vehicle simulators on a common network. It was called distributed because the simulators which supported the simulation were not physically restricted to one location but rather operated on both local and wide area networks. The formalized agreement between the content and format of messages transmitted and received on the network by the simulators was called the simulation protocol (14:1). These protocols were primarily developed to support armored vehicle simulations within the SIMNET environment (7:128).

"The primary mission of DIS is to create synthetic, virtual representations of warfare environments for the purpose of practicing warfighting skills ... when cost, safety, environmental and political constraints do not permit the field training and testing necessary to maintain combat readiness(28:3)." DIS networking protocols evolved from the original SIMNET protocols as the concepts were extended to include all classes of vehicles, consisting of high performance aircraft, surface and sub-surface naval vessels, and even satellites. It is interesting to note that the emerging standard for the DIS protocol is very similar to the original SIMNET protocol despite the extension to vehicles with wider ranges of maneuverability. (Reference (18:13-15) provides an overview of the similarities and differences of the SIMNET and DIS protocols.)

2.1.2 DIS Objectives. Principles of the emerging DIS standards and their applications are introduced in this section. Basic architectural concepts include (23:2):

1. No central computer controls the entire simulation exercise;
2. Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities;
3. A standard protocol is used for communicating "ground truth" data;
4. Changes in the state of an entity are communicated by simulation applications;
5. Perception of events or other entities is determined by the receiving application;
6. Dead reckoning algorithms are used to reduce communications processing.

The proposed IEEE draft in reference (23) provides a definition of each of these concepts. This implies that the *RDT* must use standard protocol formats to convey the "ground truth" of aircraft maneuvering on the Nellis ranges to other simulators. Dead reckoning algorithms must be used to predict aircraft positions and orientations so that the number of transmitted PDUs can be kept as low as possible. This reduces network communications traffic from a single simulator and increases the scale of the exercise that can be supported on a given network.

2.1.3 DIS PDUs. DIS simulators exchange information with each other by using a communications network. The messages which are used to convey the state and event information within the simulation are called PDUs or protocol data units. There are currently 27 different PDUs defined within DIS. A complete description of each of these PDUs can be found in reference (23) and can be grouped into the following general categories:

1. Entity information,
2. Entity interaction (such as weapons employment, logistics support and collisions),
3. Simulation management,
4. Electromagnetic Emissions,
5. Radio communications.

Of these 27 PDUs, only 12 of the PDUs have a fixed length. Each of the remaining PDUs has a variable length dependent upon many factors including numbers of articulated parts, numbers and types of emitters, length of audio transmissions, and amount of data. The PDUs of principal concern for the *RDT* are the entity state, fire and detonation PDUs. The entity state PDU has a minimum length of 144 bytes and conveys the identification, markings, position, and orientation of an entity. The fire PDU has a fixed length of 96 bytes and maps a shooter/target pair to a single weapons employment event. Variable length detonation PDUs (minimum length of 104 bytes) describe the final results of those weapon employments. (At some future time the incorporation of other PDUs into the *RDT* offers the possibility of merging radio communications with position and weapon event information.)

Table 1 depicts the entity state PDU as an example of the content and format of the DIS PDUs. The mapping of the Red Flag data messages to the appropriate fields within the DIS PDUs is covered in Chapter IV.

Word	Field	Byte 0	Byte 1	Byte 2	Byte 3
0	PDU Header	Version	Exer ID	PDU Type	Pad
1		Time Stamp			
2		Length		Pad	
3	Entity/Force ID	Size		Application	
4		Entity		Force	# Artic Parts
5	Type	Kind	Domain	Country	
6		Category	Subcategory	Specific	Extra
7	Alt Type	Kind	Domain	Country	
8		Category	Subcategory	Specific	Extra
9	Location	X Component			
10		Y Component			
11		Z Component			
12-13	Linear Velocity	X Component			
14-15		Y Component			
16-17		Z Component			
18	Orientation	Psi			
19		Theta			
20		Phi			
21	Appearance	Appearance			
22	Dead Reckon Parms	Algorithm	Unused		
23-24		Unused			
25		Linear Accel X Component			
26		Linear Accel Y Component			
27		Linear Accel Z Component			
28		Angular Velocity X Component			
29		Angular Velocity Y Component			
30		Angular Velocity Z Component			
31	Entity Marking	Char Set	Marking		
32-33		(Marking continued)			
34	Capabilities	Boolean fields			

Table 1. Entity State Protocol Data Unit (22)

2.1.4 Dead Reckoning. As seen within the example Entity State PDU of Table 1, the PDU contains more information than just an entity's position, identification and orientation. Dead reckoning parameters are also included and are used to extrapolate or predict an entity's location and orientation at some point in the future. Inputs to the dead reckoning algorithm are an entity's current location and orientation and the dead reckoning parameters from the most recent Entity State PDU. An Entity State PDU is not transmitted at the update rate of the originating simulator. Instead, a PDU is only generated when the discrepancy between an entity's current position and/or orientation exceeds a predetermined threshold(7:128).

To illustrate this concept, consider an aircraft traveling at a constant velocity on a fixed heading. If Entity State PDUs are generated to describe the motion of this aircraft without dead reckoning, then at each new time step of the simulation a new PDU must be constructed and sent. If dead reckoning is used, only the first PDU needs to be created and broadcast. As long as the aircraft remains in stable flight at the same parameters and does not deviate from its flight path, another PDU need never be sent from the originating simulator. This creates a problem, however, for simulators that connect to the network after the first PDU is sent. These simulators do not know about the existing aircraft and thus generate an incorrect image. This violates the concept of "ground truth." To overcome this inconsistency, simulators are required to generate a PDU at a minimum rate of one every five seconds. Thus the the greatest time delay incurred by any newly connected simulator would be five seconds before the "ground truth" about all entities within the simulation is received(23:24).

As one of the basic concepts underlying the DIS architecture, dead reckoning is important because it reduces the number of PDUs which must be broadcast for a given aircraft and minimizes network traffic. Research on a single F-16 aerial demonstration profile showed that network traffic could be reduced by as much as 83 percent over continuous postion updates. Dead reckoning also diminishes the computational processing associated with receipt of each new PDU because fewer PDUs are received (7:128).

Dead reckoning is a tradeoff among three factors: network traffic requirements, computational power, and the precision of the entity's location and orientation. The two factors which control these tradeoffs are the choice of the dead reckoning algorithm and the parameters which dictate when a PDU is generated. These parameters, or thresholds, represent the maximum allowable difference between an entity's predicted versus actual position/orientation. A low threshold implies that only small deviations from the entity's actual position are allowed before a PDU is generated. This may require *more* PDUs to be transmitted as an entity maneuvers outside the threshold of its predicted position/orientation; consequently network traffic increases. A higher threshold or larger difference between the predicted and actual position/orientation may result in *fewer* PDUs transmitted onto the network because of fewer deviations from the predicted movement. This reduction decreases network traffic. The benefits of network bandwidth reduction made possible by dead reckoning are not free. There is a significant difference in the computational cost between performing a position update (copy to memory) and applying the dead reckoning equations to calculate a new position(7:129).

Harvey points out that the dead reckoning concepts, which were developed for use in the SIMNET armored vehicle environment, apply equally to the DIS environments of high speed aircraft and space vehicles (7:129-130). The rate of change at which a vehicle deviates from the steady state condition (acceleration) is the determining factor for generating new PDUs. The majority of movements for both types of vehicles falls within the steady state conditions and are accurately predicted by the algorithms. Harvey also a ,ues that "there is much similarity between the rates at which armored vehicles and high performance aircraft can change their orientations and direction of travel (7:130)." Turn rates generated by slow moving tanks and fast moving aircraft are comparable and are similarly suitable for dead reckoning calculations.

The DIS protocol requires that dead reckoning be performed on both ends of the communication network (23:23-24). A simulator propagating an aircraft model along a given path in its environment maintains a dead reckoning model of the aircraft movements and transmits a PDU only when the difference between the acutal position and the predicted position exceeds the thresholds. The receiving simulator updates the aircraft position as a new PDU arrives. Otherwise the position and orientation of the aircraft for the remainder of the time is computed using the dead reckoning algorithms and parameters.

The DIS standard defines eight dead reckoning models to be used in distributed simulation. Each of these models is a unique combination of the dead reckoning algorithms. Sheasby provides a summary of the algorithms and models used in DIS (18:18-19). Because angular and linear accelerations are not available in the RFMDS data, only zero-order orientation and zero/first-order position dead reckoning are available for use by the RDT. Zero-order orientation and position dead reckoning simply means that the location/orientation is maintained constant over the frame interval (18:18-19). First-order position dead reckoning consists of integrating linear velocity over the frame interval. The following equations are used (18:18):

$$x' = x + V_x t \tag{1}$$

$$y' = y + V_y t \tag{2}$$

$$z' = z + V_z t \tag{3}$$

The use of these algorithms for the *RDT* constitutes the linear dead reckoning model. Regardless of the dead reckoning model and the specific algorithms which comprise the model, a definite decrease in network traffic can be expected.

2.1.5 Survey of DIS Applications. A movement from the SIMNET protocols to the emerging DIS standard began in 1992. A number of DIS applications under development are furthering the industry's knowledge of this new protocol and its suitability for real-time, large-scale distributed simulation. This section highlights the efforts of a handful of organizations, some affiliated with AFIT, that are developing DIS applications to contribute to this effort.

AFIT is one of several academic institutions involved in researching the capabilities of distributed simulation and virtual reality. The *RDT* and each of the applications described in Chapter I (*Virtual Cockpit, Synthetic Battle Bridge, Satellite Modeller*) rely on a common DIS entity object manager to monitor and transfer information to/from the network.

The Naval Postgraduate School (NPS) in Monterey, California has developed a real-time, workstation based, 3D visual simulation system utilizing DIS protocols. The system, called *NPSNET*, displays vehicle movements over the ground or in the air. The project centers on the development of public domain graphics simulation software which utilizes off-the-shelf hardware (19:2).

The *Virtual Cockpit, Synthetic Battle Bridge* and *NPSNET* DIS applications were demonstrated in August of 1993 at the Association for Computing Machinery's (ACM) Special Interest Group for Computer Graphics (SIGGRAPH) convention at Anaheim, California. Local Ethernet connections between two separate booths at the convention and a T-1 line to ARPA's simulation center at Arlington, Virginia demonstrated heterogeneous DIS applications interacting in a common environment.

The Institute for Simulation and Training (IST) at the University of Central Florida (UCF) in Orlando has also developed a PC-based *Computer Generated Forces* testbed that incorporates the DIS standard. The testbed consists of two major components: a *Simulator* and an *Operator Interface.*

The *Simulator* is a tool that is used to create and control semi-autonomous forces (SAF). A "human commander" provides the goals and objectives for the SAF entities in the form of keyboard or script files and the *Simulator* makes the choices for entity movement, route planning, obstacle avoidance, and target engagement. The *Operator Interface* is a high level graphic interface to the *Simulator.* Instead of using keyboard or script files, mouse-driven command menus greatly simplify control of the underlying *Simulator.*

Many of these DIS applications have been sponsored by the Department of Defense for use in military training environments. Growing emphasis in each of the military services as well as continued support from ARPA will no doubt continue to foster the development of numerous DIS applications in the coming years.

2.2 Computer Communications

A series of communication protocols is in use at various levels below the visibility of DIS application programs. Because of their complexity, these protocols are designed in

Application
Presentation
Session
Transport
Network
Data Link
Physical

Table 2. OSI Model

layers to make implementation more manageable. This section gives a brief synopsis of the protocols used in the different levels of the Open Systems Interconnection (OSI) Reference Model and their correlation to *RDT*.

The International Standards Organization (ISO) developed a guide, not a specification, for an international framework in which standards could be developed for open communications between computers. The OSI model is the result of that proposal and was developed between 1977 and 1984 (24:174). The seven layers of the OSI model are shown in Table 2. The principles used to derive the seven layers are:

1. Layers are created where different levels of abstraction are needed.
2. Each layer performs a well defined function.
3. The functions of each layer are chosen with an eye toward defining internationally standardized protocols.
4. The layer boundaries minimize the information flow across the interfaces.
5. The number of layers is a trade off between architectural complexity and distinct functionality (25:14).

For a comprehensive explanation of each of the seven OSI layers and their functions see Tannenbaum (25). In terms of the *RDT*, it is important to understand that each of the layers provides a well-defined interface between the layers directly above and below it. This has the advantage of isolating changes that occur within a given layer from any of the other layers as long as the interfaces remain the same. Protocols exist at each layer to define the interface. A protocol suite is a collection of protocols ırom more than one

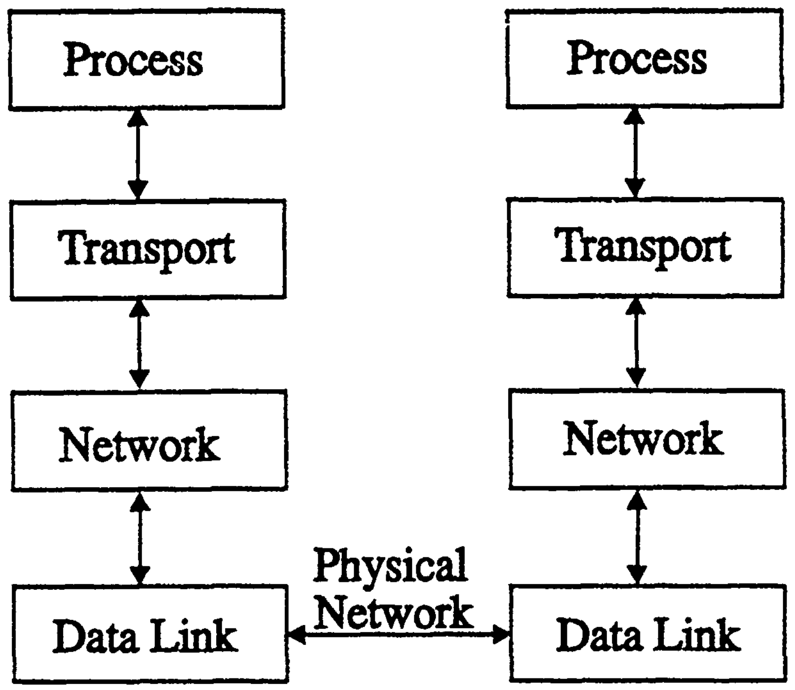

Figure 3. Simplified 4-Layer Model Connecting Two Systems

layer that forms the basis for a useful network (24:174). An example of a protocol suite, or protocol family would be the Terminal Connection Protocol/Internet Protocol (TCP/IP) suite used for communication between nodes on the ARPA network.

For explanation purposes in this thesis, the seven OSI layers have been simplified into the 4-layer model used by Stevens (24). Figure 3 shows the model for two systems that are connecteu by a network. The *process layer* consists of the top three layers of the OSI model – the *session, presentation* and *application layers.* Application programs exist at the *process* layer. The bottom two layers of the OSI model are combined into the *data link* layer. The *data-link* layer combines the network and hardware characteristics into a single, simplified layer.

The example portrayed in Figure 4 shows two hosts using DIS and the User Datagram Protocol/Internet Protocol (UDP/IP) suite to communicate over an Ethernet network (24:176). Data that is sent from one host to the other is formated according to the DIS protocol. The dashed line between the two hosts at the DIS level does not represent a physical connection but rather an abstraction. The actual flow of data goes down from the

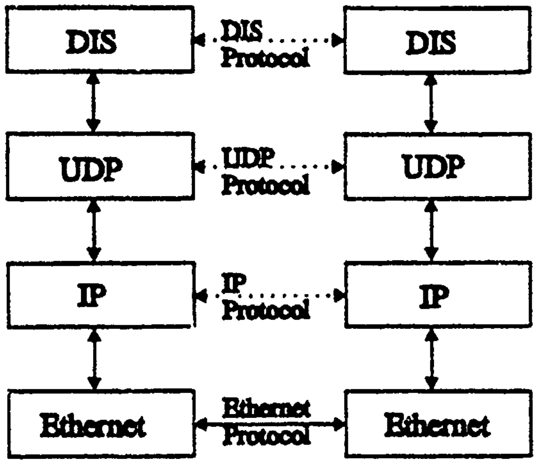

Figure 4. 4-Layer Model for DIS Application over UDP/IP on an Ethernet

DIS/*process* layer through the UDP/*transport* layer to the IP/*network* layer. The *network* layer passes the data through to the Ethernet/*data-link* layer for communication to the other host. At the receiving end, the process is reversed going up from the *data-link* layer to the *process* layer of the DIS application. At each step down through the layers, additional header information is attached to the original data so that the actual data sent contains the original DIS PDU, and a UDP header, IP header, and Ethernet header. These headers are stripped on the receiving end as the PDU moves up through the layers. When the PDU finally arrives at the *process* layer all header information has been removed and only the DIS PDU is presented to the application.

Figure 5 depicts the standard format for an Ethernet frame or packet and shows how the DIS PDU is packaged in the Ethernet protocol at the lowest layer. Note that the length of the data in a single Ethernet frame may not exceed 1500 bytes (25:145). In situations where the length of the data exceeds 1500 bytes, the data must be segmented or fragmented between multiple Ethernet packets and reconstructed at the receiving end by the IP layer. This simple example is particularly relevant to DIS applications, specifically *RDT*, as this standard dictates the use of UDP/IP protocols for broadcasting PDUs onto a distributed network.

Data transfer from the RFMDS to a RFMDS/DIS translator and out onto a network involve communications at all layers over multiple machine architectures. The OSI Model

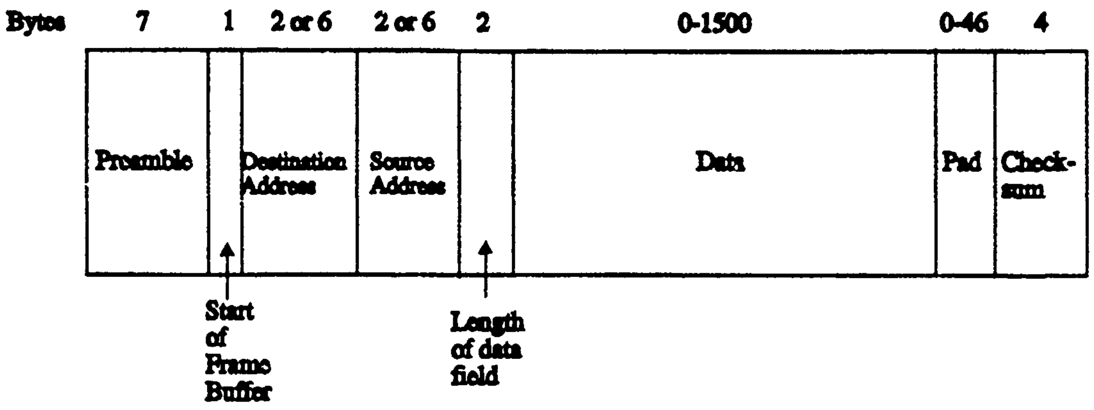

Figure 5. Ethernet/802.3 Frame Format

guidelines and the standardized hardware and software implementations of the computer industry are the key to successfully transmitting RFMDS telemetry to simulator applications.

2.3 Red Flag Measurement and Debriefing System Description

This section describes the RFMDS and provides the background information needed to understand how the *RDT* interfaces with RFMDS. This includes the system elements required to transform a modern air battle into digital telemetry data, the displays available to an aircrew member to help reconstruct and accurately debrief a mission, as well as the formats of the RFMDS data and overall RFMDS capabilities.

2.3.1 System Overview. In the deserts of Nevada and California lie the Red Flag exercise ranges managed by Nellis A[illegible] This area measures approximately 150 x 60 nautical miles and encompasses roughly [illegible]0 square miles (16:34). It is on these ranges that the mock battles of the Red Flag missions take place and aircrews learn the tactics that may one day keep them alive and ensure mission success. The RFMDS provides the technological tool to monitor, in real time, up to 136 aircraft participating in an exercise on the range. The RFMDS is capable of full state vector tracking of up to 36 aircraft equipped with specially instrumented pods. These aircraft are referred to as high-activity aircraft. Position tracking of up to 100 other aircraft is accomplished by sharing radar

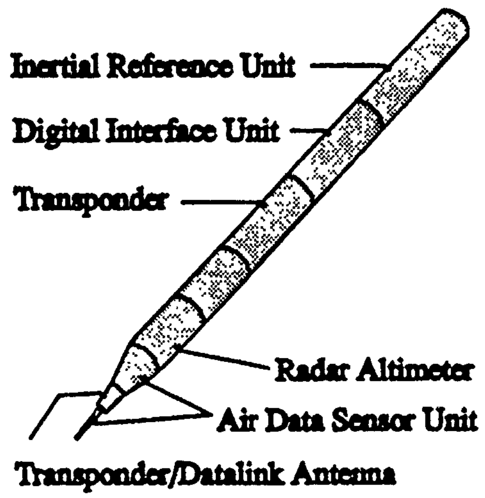

Figure 6. AIS Components

data from Federal Aviation Administration (FAA) gap-filler radars located on the range complex.

In addition to position information about aircraft on the range, the RFMDS also interfaces with up to 30 threat emitter simulators and simulates the firing of up to 50 simultaneous missiles and dropping of 22 unguided bombs for each high-activity aircraft (16:36). The elements required to bring all of this data into a central repository and present it to aircrews is the job of the four main elements of the RFMDS. These four major system elements are the Aircraft Instrumentation Subsystem (AIS), Tracking Instrumentation Subsystem (TIS), Control and Computation Subsystem (CCS) and the Display and Debriefing Subsystem (DDS). Each of these subsystems is described in the following subsections.

2.3.2 AIS. The AIS is an externally mounted, 5-inch diameter pod physically similar to the Sidewinder (AIM-9) missile. As shown in Figure 6, the AIS pod contains a transponder, a digital interface unit, a radar altimeter, an inertial reference unit and an air data sensor. The AIS interfaces with the aircraft systems to determine the aircraft's flight parameters and weapon systems status. Flight telemetry and weapons data are then transmitted over a bi-directional radio datalink to TIS ground facilities for real-time processing (22:2-6).

2.3.3 TIS. The TIS consists of two unmanned master stations and a total of 19 remote interrogator stations. These stations permit the exchange of data between ground-based main frame computers at Nellis AFB and the AIS pods. Figure 7 depicts the primary range airspace and the locations of each of the master and remote stations. These stations are positioned to provide full coverage of the range airspace from 500 ft up to 60,000 ft. Certain areas of the range have altitude coverage down to 100ft. The remote stations depicted in Figure 8 relay master station transmissions to the 36 high-activity aircraft, and in turn, relay AIS air-to-ground transmissions back to one of the master stations (22:2-8).

Each master station consists of a computer, microwave datalink, UHF radios and calibration equipment. The computer is used to process communications, measure aircraft positions, and calibrate TIS equipment. The datalink is the conduit for transmitting the telemetry information back to the CCS at Nellis.

UHF radios provide the two-way communications between pilot and mission safety observers/controllers in the Red Flag building. The calibration transponder, similar in function to an AIS pod, enables the RFMDS to do calibration and performance checks without aircraft on the range. One of these checks, called a 255, causes each master station to transmit its position back to the CCS as it is computed by triangulation from each of the remotes. RFMDS displays then depict two stationary aircraft at each of the master site locations.

2.3.4 CCS. The Control and Computation Subsystem is the primary processor of RMFDS data and uses Perkin-Elmer mainframes to support communications between RFMDS subsystems and to record system/mission data in real time. In addition, the CCS computes the state vectors for each of the high-activity aircraft by processing range measurements from both the TIS and AIS subsystems. The CCS computers also monitor maneuvering aircraft to determine whether they exceed preset limits for acceleration, descent rate, angle of attack or airspeed. The final major activity performed by the CCS is to compute weapon simulations which predict the flight paths and results of missiles/bombs employed by high-activity aircraft. All of this data is then formated and transmitted to the display subsystems for real-time monitoring (22:2-10).

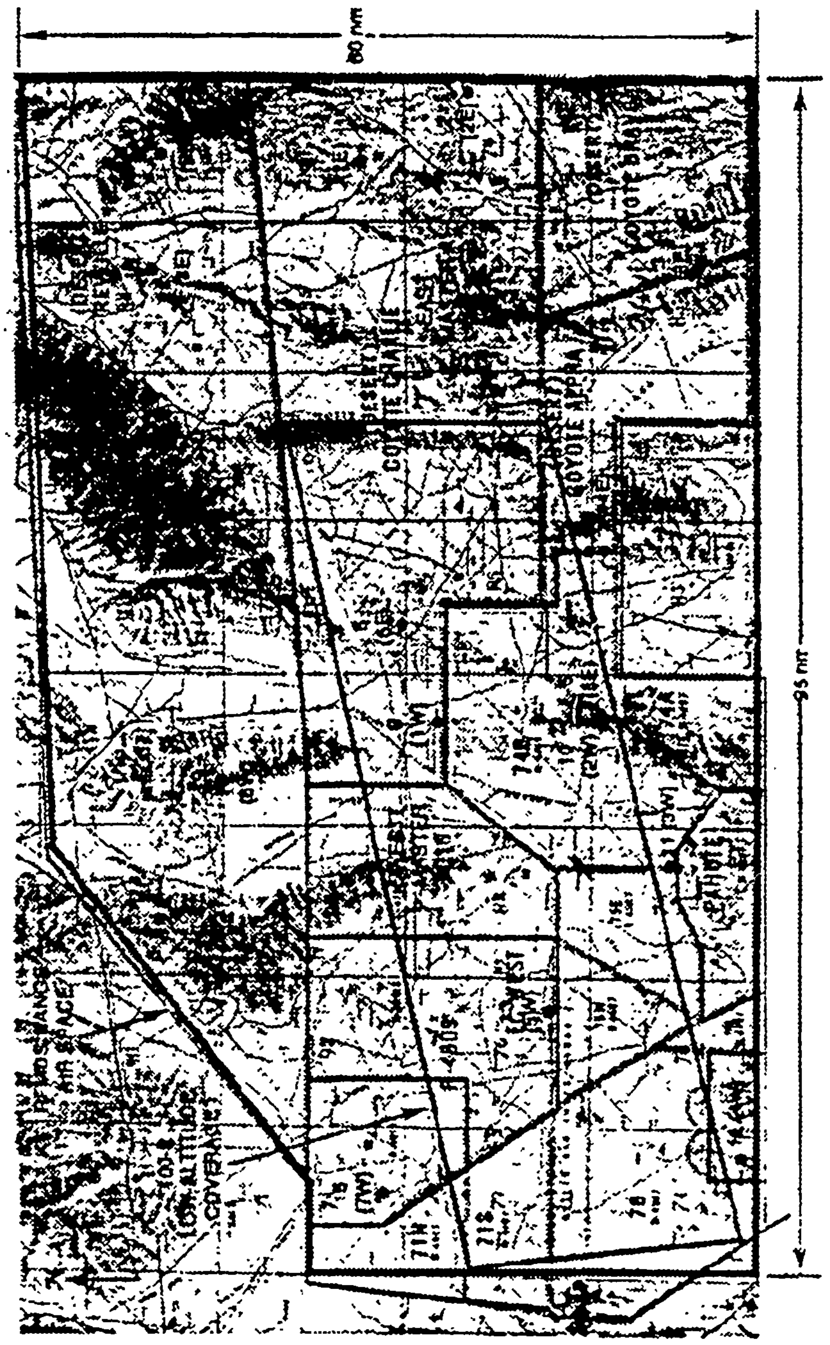

Figure 7. RFMDS Range Setup

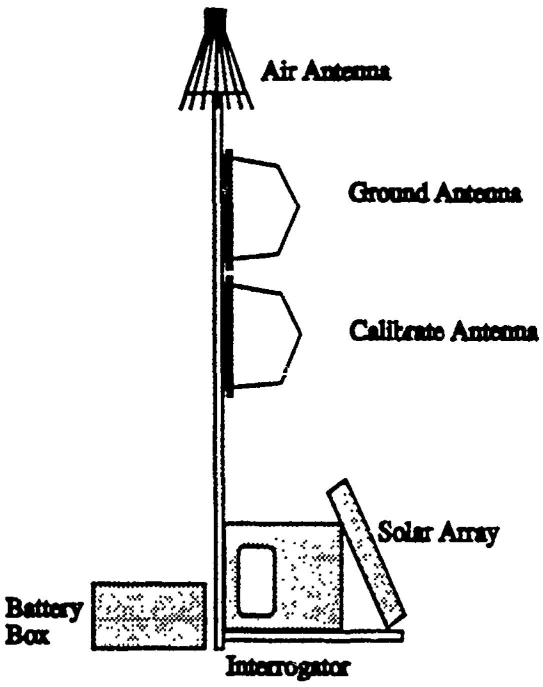

Figure 8. TIS Remote Station Assembly

2.3.5 DDS. The Display and Debriefing subsystem selectively records data received from the CCS during live operations on magnetic disk media for post-flight analysis. This data can be displayed on one of several DDS consoles or optionally projected onto large viewing screens located in the debriefing rooms or main auditorium. Up to eight channels of audio, UHF radio communciations, can be monitored and recorded as well.

Each of the DDS consoles contains three, full-color monitors, which present both graphic and alphanumeric data. The graphic images generated on the displays are of two general types, either two-dimensional (2D) or three-dimensional (3D). The 2D display, sometimes referred to as the "God's eye view," or plan view, shows the range and aircraft silhouettes from a vantage point high above the range.

Figure 9 shows a simplified version of the plan view. High-activity aircraft are depicted with 2D line drawings and low-activity aircraft are depicted with a triangle representation. Aircraft identification and altitude information is displayed next to the aircraft icon. Range boundaries are drawn and give accurate indications of the position of each aircraft at all times.

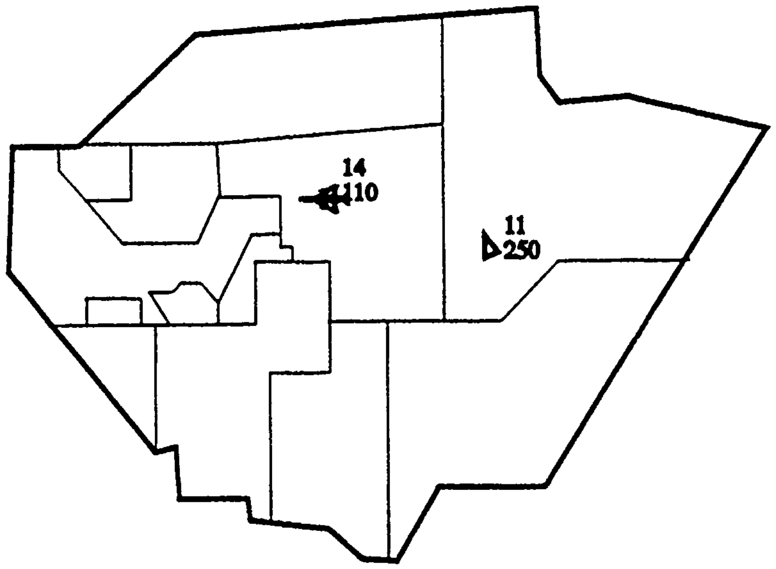

Figure 9. DDS Plan View (Simplified)

A number of options exist on the 3D displays; however, only those displays commonly used during a flight debrief are described in this thesis. One of the primary options, the centroid view (Figure 10), portrays aircraft, ground threats, targets, and simplified terrain features in a 3D viewing window. This display can be centered on any aircraft, threat or pairing of the two. Additionally, the view can be rotated 360 degrees about the vertical axis or tilted to present a vertical representation of the range airspace, which is particularly useful to show aircraft altitude separation.

A second option available on the 3D display is the cockpit view. This view shows the out-the-window scene from the point of view of any of the high-activity aircraft. Again, simple line drawings are used to render the horizon, sun, aircraft in the field of view and the cockpit outline. Figure 11 depicts an F-16 in a descending left hand turn. Infrared and radar missile cockpit cues are also available in this 3D display.

The third CRT monitor displays digital data in tabular alphanumeric formats. The data covers every aspect of range operations, from aircraft and threat monitoring to equipment status and testing. Only the displays used for air combat debriefing are highlighted. The Exercise-Data display describes who the aircraft and threat participants are and the roles that they will play during the exercise. The Pilot-Data display shows the various

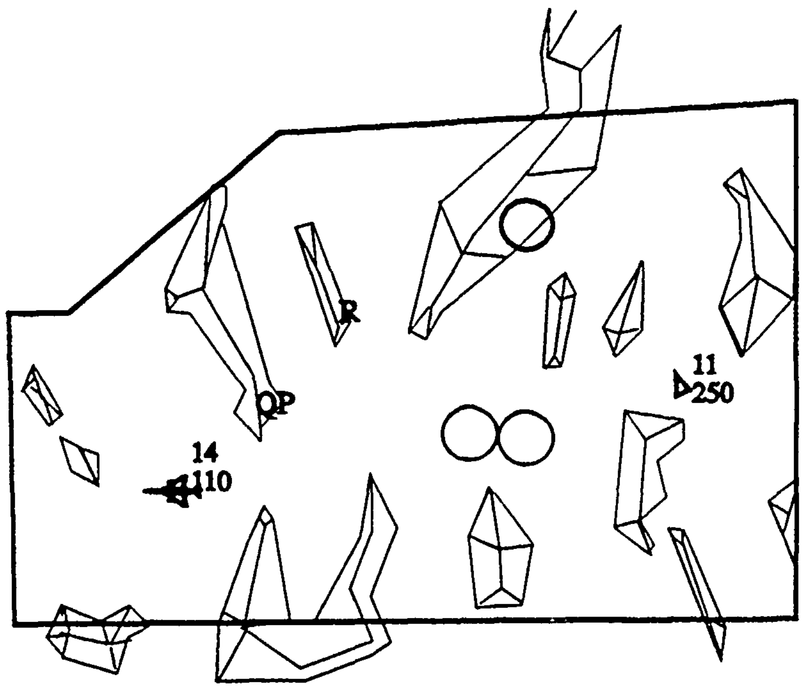

Figure 10. DDS Centroid View (Simplified)

flight parameters that would be available to a pilot in the cockpit, such as airspeed, angle of attack, pitch, roll, heading and altitude. The Quick-Look display provides selected mission and exercise data for overview of mission participants. The Summary-Data display enumerates the weapon firings and results for all participating aircraft and threats (22:4-1 – 4-68). In order to provide any of these displays, data must be transferred from the CCS to the DDS where the images can be generated. The contents and formats of the CCS messages are described in the next section.

2.3.6 RFMDS Messages. The CCS communicates with the DDS over a dedicated data link at an aggregate rate of 1.344 megabits/second. Every 100 milliseconds the CCS and DDS transmit data back and forth in order to maintain synchronization and to update the status of all participants. This translates to position updates being broadcast to the DDS image generators 10 times every second(3:A-5) and allows smooth animation of all the aircraft on the DDS displays. A 10 hertz update rate also provides sufficient data to perform an in-depth engineering analysis of both aircraft and weapon system performance.

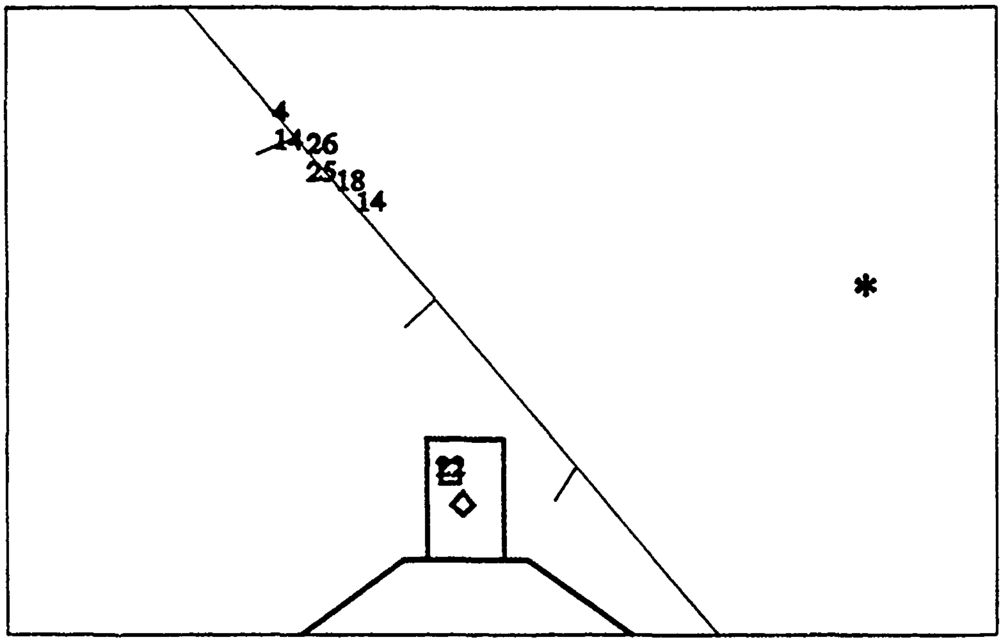

Figure 11. DDS Cockpit View (Simplified)

In the current implementation of RFMDS there are 34 different message types defined for the CCS/DDS interface(3:A-1). These messages relay information about the following general areas:

1. Bomb, missile and target status
2. Countermeasures
3. Radar information
4. Integrated air defense status/control
5. Participant identification and maneuver data
6. Range time
7. Radio transmissions
8. Electronic warfare weapon data.

Messages sent to the DDS in each 100 msec time slice are aggregated into a single fixed-length buffer with the range time message first. After all messages for the time slice

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	Message label	Blocks	Words	
1	Mission Id	Color	A/C Number	A/C Type
2	Pod ID		EW Designator	AII
3	Spare	Mission Code	Logical Player	
4-5	Aircraft Tail Number			
6-7	Pilot Squadron or Call Sign			
8-10	Pilot Name			
11	G Limit	AOA Limit	IAS Limit	
12	Pod Type	A/C Group	Descent Limit	
13	Weapon Type (1)		Weapon Type (2)	
14	Weapon Type (3)		Weapon Type (4)	
15	Weapon Load (1)		Weapon Load (2)	
17	Spare			
18-648	Repeat words 1 - 18 for up to 36 aircraft			
649	Vertical Parity Word			

Table 3. High-Activity Participant Data Message

have been transferred to the buffer, the contents of the buffer are transmitted to the DDSs. The buffer is then cleared and made available for another cycle. The contents of each of the message types varies dramatically; however, each shares a common format for header information that is used to decode the contents of the message.

Tables 3 and 4 show examples of the high activity participant and high activity maneuver data messages respectively. Common to both are the message label, the number of blocks and the number of words in the message. The message label identifies the message type and is in the range from 1 - 34. The number of blocks shows the number of aircraft for which information is provided. The number of words indicates the total length of the message(3:A- 42). For each of the 36 possible high activity aircraft in a maneuver data message (type 3), words 1 to 22 are repeated and the total length of the message is 3176 bytes. The messages of Tables 3 and 4 together provide all of the available and necessary information about the identity, position, orientation and status of an exercise participant.

2.3.7 Hardware Communications Configuration. The CCS is configured with five communication ports to which the dedicated data links may be connected. Currently,

Word	Byte 0	Byte 1	Byte 2	Byte 3
0	Message Label	Blocks	Words	
1	Spare	Status Bits	Weapons Bits	Hazard Bits
2	IR Tone	Interrogator	ITrace	Aircraft Number
3	Range Position X Coordinate			
4	Range Position Y Coordinate			
5	Range Position Z Coordinate			
6	Orientation/Heading			
7	Orientation/Pitch			
8	Orientation/Roll			
9	Angle of Attack		Angle of Sideslip	
10	Dive/Climb Angle		Pilot Yaw	
11	Normal Acceleration		Mach	
12	True Airspeed(Knots)		Indicated Airspeed (Knots)	
13	Rate of Climb		X Velocity	
14	Y Velocity		Z Velocity	
15	Inventory (1)		Inventory (2)	
16	Inventory (3)		Inventory (4)	
17	TD Radar Azimuth		TD Radar Elevation	
18	TD IR Seeker Azimuth		TD IR Seeker Elevation	
19	Pulse Repetition Freq Tone		Direction Finding Elevation	
20	Optimum A/C Tgt	Weapon Selected	Radar Mode	LGB Designated
22	Spare			
23-792	Repeat words 1 to 22 for up to 36 aircraft			
793	Vertical Parity Word			

Table 4. Maneuver Data Message

three DDS stations are attached to ports one through three. Port four is connected to a VAX computer that is used to prepare hard copy printouts of mission elements for later distribution to aircrews for post mission analysis. Port five is not normally connected; although it has the additional capability to extract selected messages from the CCS data stream under program control.

Because port four is not connected to · DDS, it is configured such that acknowledgements from the VAX computer are *not* required. This forms a one-way data stream from the CCS to the VAX, making port four the optimum point for a transparent connection of the *RDT* to the RFMDS. All information that is transmitted to the DDSs is sent to the VAX when port four is turned on except digitized audio data. This means that another audio data source must be found if the eight channels of radio communications are to be made available to the RDT.

2.4 Performer

Inherent in any visual simulation application is the need to reduce the complexity involved in developing the image generation and improve the performance or rate at which the images can be displayed on the computer screen or display device. Because the RDT shares many of the attributes of a visual simulation, these same needs apply. Silicon Graphics, Inc. has attempted to solve some of the complexity associated with developing visual simulations with their *Performer* library. This library is an effort to create a standard set of procedures specifically designed to meet the needs of visual simulation. This section provides a brief overview of *Performer*, and the *Performer Programmer's Manual* is recommended as the definitive reference.

Performer is a library of C-callable routines that create a new interface to the SGI graphics pipeline. This interface vastly improves a number of areas of the graphics library (GL) interface that comes standard with each SGI platform or machine. The *Performer* library removes many of the chores associated with programming the SGI, and captures the corporate expertise of SGI for programming their hardware pipeline into practical routines.

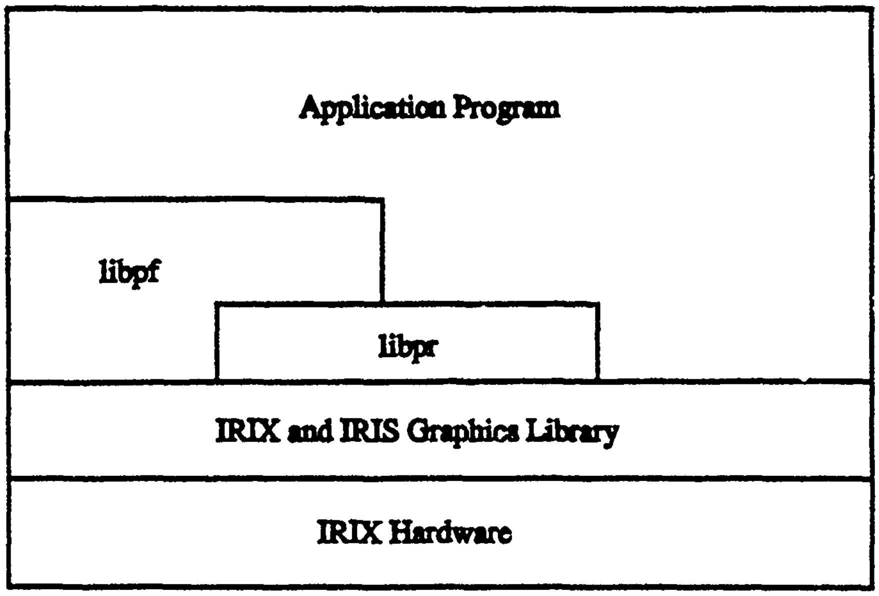

Figure 12. IRIS Performer Library Hierarchy

The main components of the *Performer* toolkit are two libraries. One library, *libpr*, provides optimized low–level rendering functions, state control, and other fundamental real time graphics functions while the other library, *libpf*, provides a visual simulation development environment that layers multiprocessing, database traversal and rendering on top of the *libpr* library. Figure 12 illustrates the relationship between an application program, the performer libraries, the IRIS Graphics Library, and the IRIX operating system (11:2-1). As the figure indicates, the application program has direct access to all of the performer library functions as well as free access to the graphics library and operating system functions. This allows general development within the performer framework and specialized development with the graphics libraries as required.

Performer maintains an internal data structure, called the geometry tree, which is traversed during the rendering process. This tree contains the common geometry types used on the SGI. Geometry nodes, or geodes, can hold polygons, triangle meshes, light points, and many other types of geometric primitives, and are the leaves in the geometry tree. The tree also holds additional information that is useful in changing the characteristics and presentation of the final image. Examples of the types of additional information

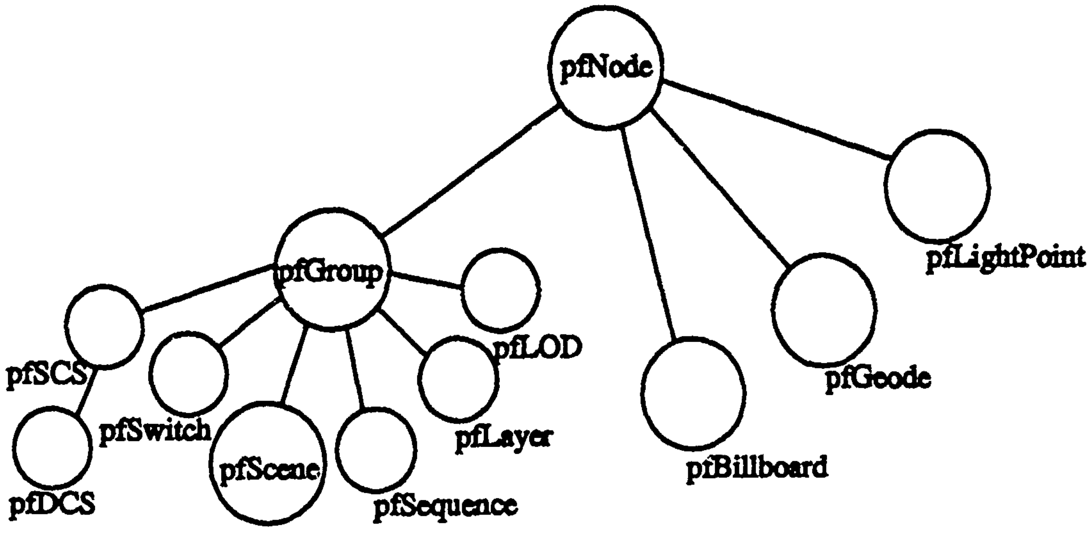

Figure 13. IRIS Performer Node Hierarchy

are: switching distances and models to be used when a different level of detail is desired, animation sequences, and the types of coordinate system used by the various geodes.

Because the geometry tree is not dependent on a particular external geometry format, any format which can be converted to the *Performer* internal geometry format can be employed in the tree. This conversion is done through file reader routines that are used with one of three different geometry formats– *Multigen* '.flt' format, SGI '.sgi' format and a '.bin' format. The *Multigen* .flt reader allows all of the power and hierarchy expression available with the *Multigen* modeling tool, which makes it ideal for use in *Performer* applications with minimal effort.

Performer also maintains state information in the tree. These nodes, called geostates, hold details about materials, textures, transparency, lighting conditions, and other facts which are normally part of the GL state. This allows *Performer* to easily render multiple textures, transparency, and effects, through its built-in state management features. Figure 13 depicts the hierarchy of different nodes that can be used to build an application database (11:5-2). More complete information about each of these nodes and their properties can be found in the documentation supplied with the *Performer* software.

Performer renders its geometry tree very efficiently. According to Silicon Graphics, *Performer* maximizes frame rates for the SG geometry pipeline. This implies that the limiting factors for efficiently rendering a set of geometry are dependent on the complexity of the geometry and scene management, not on rendering code efficiency, when *Performer* is used.

Another breakthrough in *Performer* is obtained through its multiprocess management features. *Performer* provides an abstract, easy to use model for using up to three processes on a multi-processor machine. This model dedicates one process to drawing, one to culling, and one to the application managing the scene. (Culling is the process of removing geometric objects from the scene because they are not visible within the current field of view.) These processes are then distributed among the available processors and communicate with each other through shared memory. This feature is also extensible to machines with more than one rendering pipeline.

Other features in *Performer* include multiple channels (viewports) into a scene, an easy to use viewing model, and an extensive math library. *Performer* has built-in collision detection features, intersection testing, and special effects processing (fog, haze, time of day, earth-sky, etc.)

All of these features and routines combine to provide a libıary of computer image generation software that is adaptable to applications in the visual simulation environment and specifically to the development of the RDT. Chapters 3 and 4 build upon this background information to construct the designs for the computer communications, translation and image generation functions of the *RDT*.

III. Computer Communications and Data Translation Design

This chapter presents a pattern, or design, for creating two major components of the *RDT*: the ccmputer communications component (CCC) a .d the data translation component(DTC). Issues related to the user interface and visualization component (IVC) are addressed in Chapter IV. A description of each of these two components begins by first stating the objective or functional goal that the implementation must achieve and then defining the operating characteristics or constraints under which the system must operate. Next, possible approaches to accomplishing the design objective are enumerated along with their advantages and/or disadvantages. The reason for selecting one approach over another for the final design is also indicated. Finally, each component description details the algorithms, methods and devices chosen to carry out the plan of attack.

3.1 Computer Communications

The two principal objectives of the computer communications component (CCC) are to (a) install a transparent device into existing RFMDS data lines to create a one-way data stream that can be used to monitor data messages sent from the CCS to each of the DDS computers; and (b) reformat and broadcast the data messages as Ethernet packets onto a local area network for use by the DTC. (The requirement to use the Ethernet protocols in the data link layer is self-imposed because of the availablity of both software and hardware at AFIT that can be used to transmit, receive, and monitor Ethernet packets. In addition, Ethernet has a 10 megabit/sec (Mbps) bandwidth (25:144) that is easily capable of delivering the 1.344 Mbps of data sent by the RFMDS.)

3.1.1 Communications Analysis. Figure 14 represents the RFMDS hardware configuration describe. in Chapter II. The CCS and the three DDS computers are located on the ground floor of the Red Flag building in the main computer room. Ports one through three of the CCS are each connected to one of the three DDS computers. Each DDS computer controls two DDS consoles which are located in debriefing rooms throughout the building. A total of six DDS consoles are available for simultaneous use to either monitor live missions or replay past exercises for mission debriefs. The communications channels

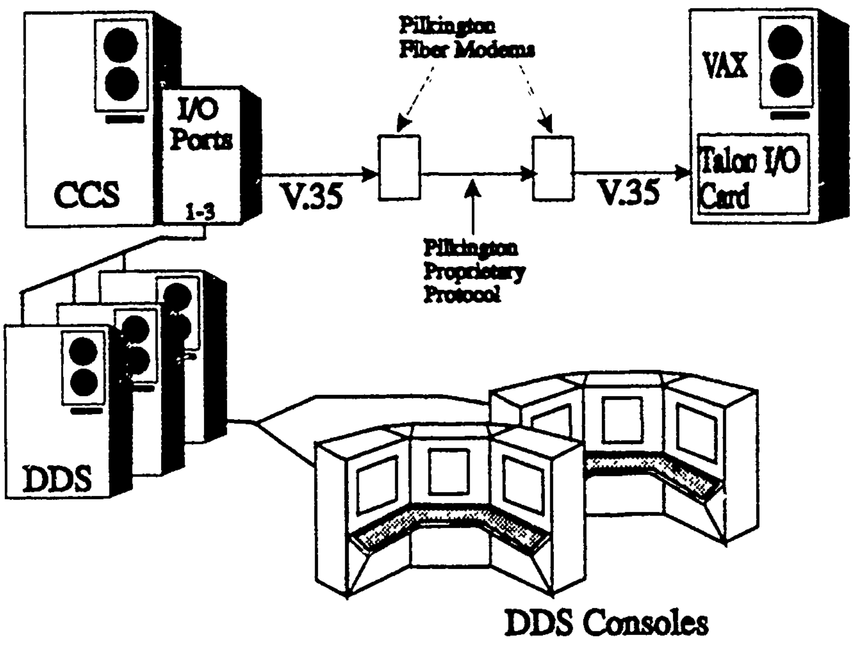

Figure 14. RFMDS Hardware Configuration

between the CCS and DDS computers are bi-directional data streams and many CCS to DDS messages require DDS acknowledgements.

The VAX computer connected to port four, is physically located in a room adjacent to the CCS/DDS computer room. Data sent to the VAX from the CCS is transmitted using the CCITT V.35 protocol using Pilkington fiber modems. The fiber modem retransmits the messages using Pilkington's proprietary protocol over a fiber optic cable to a second Pilkington fiber modem. This second modem converts the data back into the original V.35 protocol and forwards the messages to a Talon communications card within the VAX computer. Message acknowledgments from the VAX computer are neither required nor expected.

Workspace for AFIT computers located across the hall from the VAX computer in room 115/116 is a straight-line distance of approximately 50 to 75 feet. A fiber optic cable, installed by Cubic Corporation (the RFMDS developer), connects computers in room 116 to a DDS console in room 234 (one of the six debriefing rooms).

Discussions about the software programming of the RFMDS with Bill Saucier, former Cubic Corporation engineer, revealed the following essential information.

1. Messages to be sent to the VAX computer are accumulated into a buffer before being transmitted as a single fixed-size data block. (The maximum size allowed for the data block is 8000 bytes, but the full capacity is rarely used. The VAX computer is set to receive only 6144 byte blocks.)
2. The first message within each block is the range time message.
3. Subsequent messages are placed sequentially in the block until either there are no remaining messages for the current 100msec time slice, or the block is filled to capacity. These events trigger a parity calculation and the transmission of the data block.
4. A series of three or more bytes containing the hexidecimal value '0x7f' preceed the data messages in the block. A single hexidecimal byte containing '0xff' in the message type field signals the end of any further messages in the block.
5. Excess space at the end of the fixed-size block is unused.

3.1.2 Design Resolution. The hardware and software characteristics, as well as the details of the system's physical layout described above, present a number of possible approaches for the design of the CCC. This section examines each of the possibilities and presents the rationale behind the selection of the final design.

At least three possibilities exist for installing a communications monitoring device into the RFMDS data stream. The first option involves connecting directly to the unused port, number five, of the CCS. Such a connection, however, would require a software modification to the RFMDS kernel and many hours of dedicated system development time. The potential for problems introduced by modifications into an already complex system far outweigh any benefit that could be derived by using this method over other possibilities. Even the slightest inconsistency in the interface with the CCS could negatively impact the day-to-day operations of Red Flag and violate our major design objective of transparency. Connecting to port five was not considered a viable option.

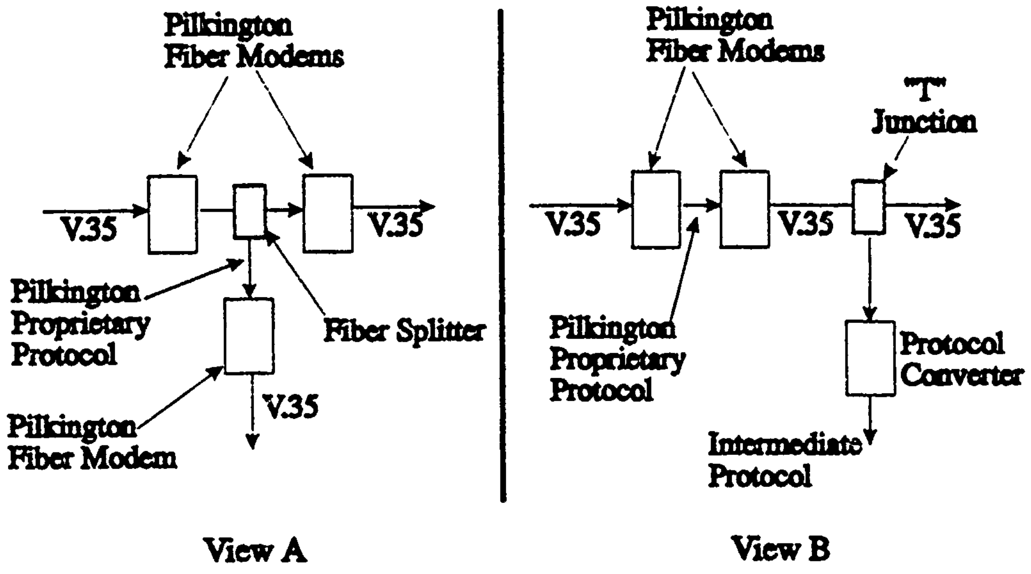

Figure 15. Possible 'T' Junction Placements

The other two possibilities both involve creating a 'T' junction into the communications link between the CCS and the VAX computer attached to port four. The difference between the two approaches is in the placement of the junction. Option one, depicted in View A of Figure 15, places the 'T' into the fiber link between the Pilkington modems. Option two puts the 'T' between the second Pilkington modem and the VAX computer.

In order for option one to be successful, a fiber splitter must be inserted between the two modems. Because the protocol used between the modems is proprietary, a third Pilkington modem at the end of the 'T' is necessary to recover the original V.35 data. Discussions with the distributers of the Pilkington modem revealed that the modems currently in place in the RFMDS are no longer manufactured and upgraded replacements are available at substantial cost. Placement of a fiber optic splitter into the fiber cable would potentially cause a reduction in signal strength at the second Pilkington modem. Information was not available about the extent of any such signal reduction and its impact on normal operations.

View B of Figure 15 shows the 'T' junction placement between the VAX and the Pilkington modem. The only protocol to be dealt with here is V.35 and the purchase of an additional modem is not required. This option offers the simplest design at a minimum cost.

Another problem arises due to the fact that CCITT V.35 is a short range protocol and is not intended for signal transmissions over medium to long distances. The 50-75 foot distance from the VAX computer to the available work areas makes the use of V.35 impractical. To overcome this difficulty, an intermediate protocol, with transmission capabilities to cover the 50-75 foot distance between computers can be used but this requires another device to make the conversion.

The search for a means to convert the data from V.35 to a suitable protocol produced only one device capable of the required 1.344 Mbps rate over the 75ft distance–a CCITT V.35 to RS-422 converter. (RS-422 is capable of transmitting up to 2Mbps over 60 meter cables (25:77).) This type of converter makes it possible to receive the data messages in room 116 over an RS-422 cable at the required data transmission speeds.

The final hurdle in establishing the transparent link is the conversion from RS-422 to Ethernet. This problem, however, does not lend itself to the use of a simple converter because of an anticipated need to save the RFMDS messages onto non-volatile storage media. In addition, preliminary examination of over 30,000 data blocks from a single Red Flag mission revealed that approximately 30 percent of the space in the blocks was unused. A significant reduction in the amount of data transmitted to the DTC can be achieved if the device used to convert from RS-422 to Ethernet can be programmed to filter out the unused data from the data blocks. These two factors dictate the use of a computer/workstation to perform the conversion, storage and filtering.

The final hardware design for the computer communications component is shown in Figure 16. It illustrates the use of a 'T' junction between the Pilkington modem and the VAX computer. Data messages in the V.35 protocol are routed to a computer for conversion to Ethernet and retransmission to the DTC.

3.1.3 Software Methods. The tasks to be accomplished by the conversion program involve reading the data from port four, parsing the blocks into individual messages and

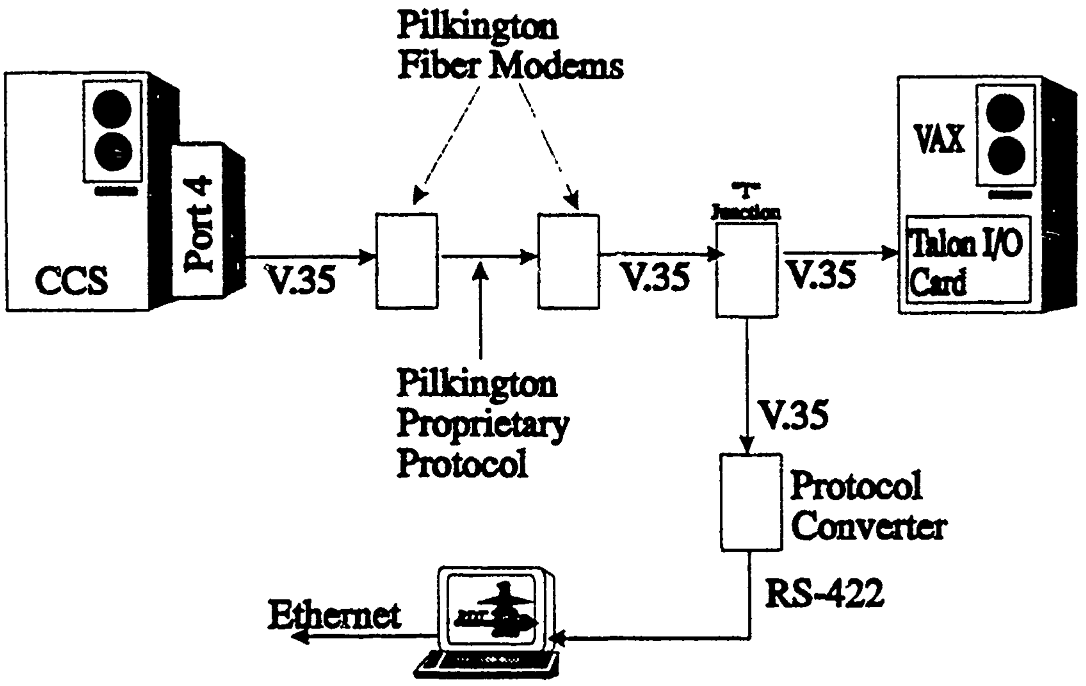

Figure 16. Final Design of the CCC

transmitting them onto a network as Ethernet packets. A simplified algorithm written in a "C"-like computer language best illustrates how these tasks can be accomplished.

```
while (Data available at the V.35 port)
{
     Search for start of message bytes (0x7f 0x7f 0x7f);
     Read data block into buffer;

     /* Parse the data block until ''end of data'' */
     while (message type not equal to '0xff')
     {
          Get next message;
          Add Ethernet header information;
          if (message length > max Ethernet packet length)
          {
               Segment message into allowable packets;
          }
          Output message to Ethernet port or
          Archive message to disk;
     } /* end inner while */
} /* end outer while */
```

The design of this algorithm causes each message type to be transmitted as an independent unit rather than dividing the data block into maximum sized Ethernet packets. This necessitates the transmission of more packets, yet it also simplifies the processing of messages by the DTC because the messages have already been extracted from the block and can be used immediately upon receipt without reconstruction and parsing. This algorithm also eliminates the transmission of unused bytes in the block because parsing ceases once the "end of block" message type is seen. The high speed at which the DTC must operate to perform the data translations creates the need to reduce the amount of processing.

3.2 Data Translation

The principle objectives of the data translation component are to receive RFMDS messages containing the aircraft telemetry from the CCC and translate them into the formats prescribed by the DIS standard. This may involve something as simple as copying the data from the Red Flag message buffer into the DIS PDU buffer; or it may involve a series of complex computations. The final objective is to broadcast the DIS PDUs using a best effort, multicast communication service. (Broadcasting messages to a group of network stations instead of a single site is called multicast communications (25:144).) During the presentation of this section a number of issues are raised which are particular to the simulation of high performance aircraft yet have no current resolution within the DIS standard. Insights gained during the design phase of the DTC may benefit others who are attempting to incorporate telemetry from live aircraft into a DIS environment.

3.2.1 Data Translation Analysis. In order for the RFMDS messages to be translated from their native frame of reference to the DIS frame of reference there are three basic ideas which must first be understood concerning the fixed formats of RFMDS and DIS.

1. The difference between the floating point number formats of RFMDS and DIS;
2. The difference between aircraft orientations in the RFMDS and DIS frames of reference;

3. The frequency of RFMDS aircraft updates for high/low activity aircraft and weapon simulations;

This subsection addresses each of these issues in turn. It should be noted however, that while the RFMDS message formats have been in place for a number of years now, the DIS formats do not enjoy the same stability. Changes to the DIS standard are constantly being proposed. Thus, issues raised here as unresolvable may be fully resolved and incorporated into some future version of the standard.

3.2.1.1 Floating Point Representation. Floating point numbers in the RFMDS messages use a different format than the *IEEE* 754-1985 floating point standard prescribed in the DIS (23:76) standard. Whereas the *IEEE* standard describes a 23 bit fraction, 8 bit exponent and a sign bit (26:570), the RFMDS floating point format uses a 24 bit fraction, 7 bit exponent and a sign bit. This nonuniformity between formats requires that each floating point number in the RFMDS message buffer be converted to the *IEEE* 754 standard format before being placed within the DIS PDU. In addition, the floating point conversions must be accomplished prior to the use of these numbers in any calculations within the DTC because the hardware and software within the SGI computers uses the IEEE standard. Conversions between the two floating point formats add a *significant* computational cost to the translation.

To get an idea of the computational cost to perform these conversions, assume that there are 36 high activity aircraft whose position is being updated. For each of the 36 aircraft there are approximately 10 maneuver data messages that are broadcast each second. In every maneuver data message, each of the 36 aircraft has 9 floating point numbers representing its location, orientation and velocity that must be converted. This requires that 3240 floating point conversions be completed each second. Algorithms designed to perform these computations use approximately 588 CPU cycles per conversion (15). This brings the total number of CPU cycles/per second (IRIS 4D/440 VGXT) which must be dedicated solely to the floating point conversions to 1,905,120.

3.2.1.2 Aircraft Orientation/Location. The cost of performing translations between the RFMDS and DIS frames of reference for an aircraft's orientation/location in a non-standard DIS "flat earth" environment is largely absorbed into the floating point calculations. (In a "flat earth" environment, the curvature of the earth is ignored.) If however, the DIS standard is strictly adhered to and the WGS84 geocentric coordinate reference system is used, then an additional significant cost is incurred in translating and rotating the location and orientation of each aircraft into the geocentric coordinates. (More information on coordinate conversions can be found in Erichsen (4).)

3.2.1.3 RFMDS Updates. As indicated earlier, the RFMDS provides position updates for each of the high activity participants at approximately 10 hertz.

This 10 hertz time differential between RFMDS blocks cannot, however, be taken as absolute. In one sample of 1,000 consecutive blocks of RFMDS data, roughly 97% of the message blocks were transmitted within .1 ± .02 seconds. The remaining 3% however were well outside the 10 hertz interval. Figure 17 shows an impulse plot for each of the 1,000 time intervals sampled. Note that the occurrence of time intervals outside of the specified 10 hertz update rate is not regular and thus not suitable for use as an absolute time increment. These irregularities suggest that dead reckoning algorithms use the time differential computed between successive RFMDS message blocks rather than some fixed time interval approach designed to minimize delta time calculations. Likewise, an approach that uses a real-time clock to compute time differentials for dead reckoning purposes will be in error roughly 3% of the time and cause erratic position updates.

Updates for low activity aircraft are sent at irregular intervals and generally exceed the 12 second maximum time interval between DIS entity state PDUs. On some occasions, update intervals between low activity aircraft have been as long as 25 seconds. This causes a problem in the DIS world because entities which are not updated within 12 seconds are to be deleted from the simulation (23:25). This should not be allowed during the replay of a Red Flag mission due to the distractions caused when low activity aircraft constantly disappear then reappear when they are deleted and reinserted into the exercise.

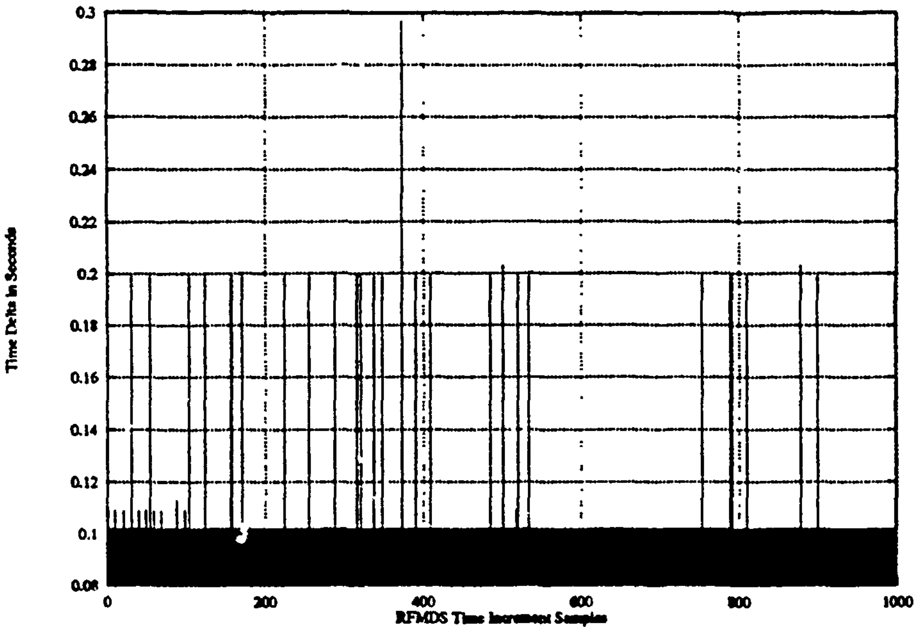

Figure 17. Time Differentials between Successive RFMDS Messages

Because one of the *RDT's* objectives is to provide a debriefing tool that can be used to portray the activities of a given Red Flag mission, it is important that the images displayed by the *RDT* represent the mission as accurately as possible. In the case of low activity aircraft where position updates occur less frequently, a decision must be made whether or not to perform dead reckoning. If dead reckoning is performed for the low activity aircraft then the position of the aircraft at time t following a position update is questionable. Consider a low activity aircraft whose position has just been updated via an entity state PDU. It begins a standard rate turn, but because another update for this aircraft may not come for as much as 25 seconds, it will appear on the *RDT* displays to continue in a straight line for 25 seconds and then abruptly move to its new position as shown in Figure 18.

If dead reckoning is not performed, another problem occurs. Consider the case where that same aircraft begins a turn and dead reckoning is not used. The *RDT* image will show only the last update without further aircraft movement. Upon receipt of the next position update 25 seconds later, the new position is displayed. Figure 19 accurately portrays a

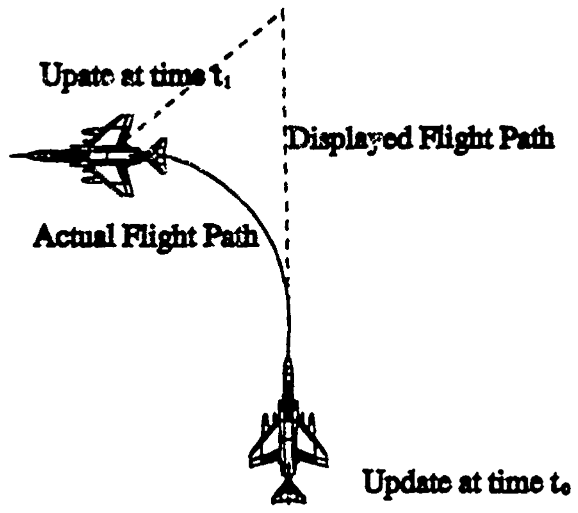

Figure 18. Low Activity Aircraft Positions with Dead Reckoning

history of actual aircraft movements, in contrast to misleading flight paths of low activity aircraft that have been dead reckoned.

Simulated weapons employed during a Red Flag exercise also have state vector information transmitted at the same rate of approximatly 10 hertz. One entire processing unit of the CCS is dedicated to simulating weapon trajectories and results for up to a maximum of 50 weapons. These updates are calculated and packed into a RFMDS message block during each 100 msec cycle that weapons are active. Unfortunately, however, only a coordinate location is generated. Weapon velocities and orientations are not part of the weapons data message and must be computed in order to provide sufficient data for both the dead reckoning algorithms and the placement of that information into the fire, detonation and entity state PDUs.

The last item that should be mentioned relates to the total number of position updates that can be expected during a typical Red Flag mission. Data collected during a 44 minute and 39 second mission flown on 15 July, 1993 contained approximately 800,300 position updates. This data includes the full flight path histories of more than 40 aircraft and numerous missile shots. If these updates were to be broadcast as entity state PDUs, this would require, on average, the transmission of roughly 283 PDUs every second for the

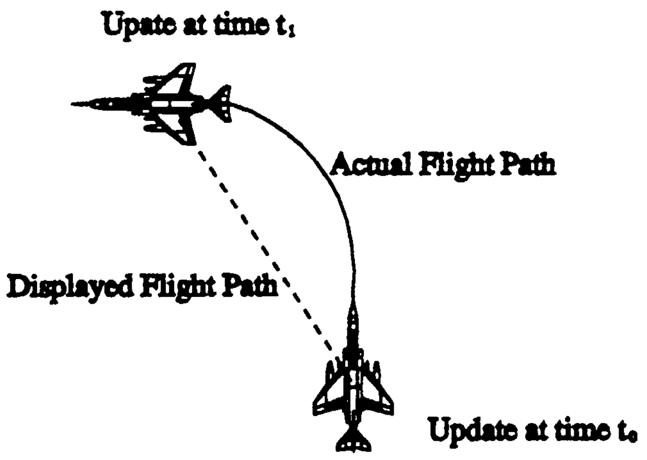

Figure 19. Low Activity Aircraft Positions without Dead Reckoning

duration of the mission if dead reckoning was not performed and this would be a significant load on an Ethernet local area network (LAN).

3.2.2 Design Approaches. Although there are many ways to construct an algorithm which performs the data translations from RFMDS to DIS, a straight-forward approach, which accepts data from the RFMDS and then translates it sequentially into DIS PDUs, is described in the next subsection. The principle issues addressed in this subsection relate to the mapping of the information contained in the RFMDS data messages to the appropriate fields within each of the DIS PDU record types. Examination of the data within the RFMDS messages suggests that the information can be grouped into the following general categories:

1. Available in RFMDS telemetry messages and directly transferrable to DIS PDUs.
2. Available in RFMDS telemetry messages but not transferrable to DIS PDUs (Data can be reconstructed from information within the PDU)
3. Available in RFMDS telemetry messages but not transferrable to DIS PDUs (Data can *not* be reconstructed from information within the PDU)
4. Required in DIS PDUs but not available in RFMDS telemetry messages

Data that is available in RFMDS telemetry messages and directly transferrable to DIS PDUs includes such items as: location, orientation, and linear velocity. Although each of these items may need to be converted to standardized formats, the information is available and can be converted in a straight-forward manner.

Angle of attack and rate of climb must be reconstructed from information placed in the DIS PDUs that is extracted from RFMDS telemetry messages. Both of these items can be calculated from the aircraft's velocity and orientation vectors and would be an unnecessary addition to information already contained within the DIS PDUs.

Examples of data that can not be reconstructed from information in the DIS PDUs but is available in RFMDS telemetry messages includes the following: true airspeed, indicated airspeed, mach number, crab angle, angle of side slip, weapons status, current selected weapon, radar azimuth/elevation, infrared tone frequency, and infrared seeker azimuth/elevation. Some of this information, for example indicated airspeed and mach number, is related to the density and temperature of the atmosphere. Atmospheric data is not currently available in the DIS protocol; and thus this information is not available to DIS applications. It might be argued that the data in this category need not be transmitted to other simulators, yet one of the purposes of the *RDT* is to provide as much of this type of information to aircrews as possible so that accurate assessments of their performance can be made. It is often the case that the radar or infrared seeker azimuth/elevation depicted on the aircraft's head-up display (HUD)in the RFMDS cockpit view is used to identify which aircraft is being targeted at the time of a missile launch. DIS protocols make no provision for this type of information. Even if this data were to be contained within the DIS PDU, it could not be used reliably because dead reckoning suppresses broadcast of the majority of the PDUs and generation of new entity state PDUs is not dependent on this dynamic data.

Examples from the final category include DIS specific items which are unrelated to the RFMDS messages. Some of these items include: version, exercise ID, time stamp, site, host, appearance, and dead reckoning parameters. Most of this data can be easily generated for inclusion into the PDUs, however, linear and angular accelerations are not found in any of the RFMDS data messages. If included, these accelerations must be calculated and stored in the PDU.

Information that falls into category 1 can be simply converted and put into the DIS PDUs. A discussion about possible options for completing this task would be pointless. Information in category 2 does not require placement into DIS PDUs and therefore requires no special procedure to make a conversion. Most of the data in category 4 can easily be generated and a discussion about ways to create and store this data in the PDUs would also prove of little value.

The task of translation of the RFMDS telemetry messages in category 3 into DIS PDUs can be solved a couple of interesting ways. One approach is to place as much of the data as possible into unused or pad fields of currently defined PDUs. This allows the data to be broadcast, but deviates from the established protocols. This approach also has the advantage of retaining the defined sizes of the DIS PDUs without causing an increase in network bandwidth. These deviations may require considerable redesign and effort if the formats of the PDUs change and the unused/pad fields, previously ignored by standard DIS applications, are now used.

A second approach calls for the creation of RFMDS-specific PDUs. These PDUs would be ignored by other applications but would be intercepted and used by the *RDT*. Although this approach preserves the formats defined within the DIS standard, it also generates a number of questions. First, how often does this information need to be transmitted? In other words, if dead reckoning suppresses broadcast of the majority of the PDUs, should these special PDUs with time sensitive, dynamic data be suppressed also? What effect will a significant number of non-standard PDUs have on the network bandwidth? Will sufficient bandwidth still be available if the scale of the exercises increase? These questions bear serious consideration and warrant further research.

3.2.3 Design Resolution. Now that the significant issues and options have been explained, this subsection presents the final functional design used to implement the DTC and the rationale behind these design decisions. In a manner similar to the description of the CCC design, a "C"-like language is used to characterize the algorithm which translates

RFMDS messages into DIS PDUs. Next, the decision to perform dead reckoning calculations for only the high activity aircraft and weapons is explained. The final portion of this section details the mappings of data from the RFMDS message formats into the DIS PDUs.

3.2.3.1 Software Methods. The tasks necessary to translate the RFMDS message formats into DIS PDUs are very straight forward and include retrieving a message and then performing the appropriate activity based upon the message type. This process can best be illustrated by the following algorithm.

```
while (RFMDS messages are available)
{
     Read message into buffer;

     /* Handle each message type individually */
     switch(message type)
     {
       case TIME:
          set range time;
          break;

       case HIGH ACTIVITY MANEUVER DATA:
          for each aircraft
          {
               perform dead reckoning calculations;
               if (send this PDU == TRUE)
                    Fill & send Entity State PDU;
          } /* end for */
          break;

       case LOW ACTIVITY DATA:
          for each aircraft
          {
               Fill & send Entity State PDU;
          } /* end for */
          break;

       case HIGH ACTIVITY PARTICIPANT DATA:
       case LOW  ACTIVITY PARTICIPANT DATA:
          for each aircraft
          {
               Replace old participant data
```

```
                        with new data;
            } /* end for */
            break;

         case WEAPONS:
            for each weapon
            {
                 if this is a fire event
                      Fill & send Fire PDU;
                 else
                      if this is a detonation event
                           Fill & send Detonation PDU;
                      else
                      {
                           perform dead reckoning
                                   calculations;
                           if (send this PDU == TRUE)
                                Fill & send Entity State PDU;
                      } /* end if */
            } /* end for */
            break;
      } /* end switch */
   } /* end while */
```

3.2.3.2 Weapons and High Activity Aircraft Dead Reckoning. Evident in the preceding algorithm is the design decision to perform dead reckoning calculations for only the weapons and high activity aircraft. The reason behind the decision not to dead reckon the low activity aircraft primarily rests with the concern to present an accurate depiction of the low activity aircraft flight paths rather than continuous, potentially inaccurate, movements. This is not a concern for the high activity aircraft and weapons because of their frequent position updates.

It should also be noted that the time interval used in the dead reckoning calculations is computed from the time between successive RFMDS range time messages and not a real-time clock in the DTC. This ensures that the time interval corresponds to the actual time interval in the RFMDS data and guarantees that delays due to operating system activities will not be introduced by using the system clock.

3.2.3.3 Data Mappings. Perhaps the simplest format for describing the final design for mapping RFMDS data into DIS PDUs is a set of figures which graphically depict where data found in the RFMDS messages is placed into the DIS PDUs. Accordingly, the next several figures represent each of the RFMDS message types that are used in the DTC and the mapping of their information into the entity state, fire and detonation PDUs. Figure 20 shows how entity state PDUs are generated for high activity aircraft from the high activity participant and maneuver data messages. Figure 21 displays the mapping of data for low activity aircraft from the low activity participant/data messages. Figures 22, 23 and 24 depict the mapping of RFMDS weapons data messages into fire, entity state and detonation PDUs.

Highlighted in the figures are the fields in the DIS PDUs that are not used or are filled with constant data. Unused fields in the PDU are due to the fact that the *RDT* represents a basic system, limited in scope, that was developed as a "proof of concept" and not a full-scale commercial product. Thus some of the information that could be transferred to the DIS PDUs and used in the image generation software, such as articulated parts fields, is not currently used, but is available for future implementations.

Fields in the DIS PDUs that are depicted with the double box represent compromises to the DIS standard. Information placed within these fields reside in pad fields or unused areas. RFMDS time stored in the first 4 bytes of a 15 byte unused field within the dead reckoning parameters is a good example of how *RDT* modifies some of the DIS PDUs to provide a capability that is not available in DIS. The Zulu, or Greenwich Mean Time, depicted by RFMDS on all of its displays is a critical component for aircrews when they debrief their missions. The times of key events, such as missile shots, are recorded by the pilots as they fly. Later during the mission debrief, these events are compared with the time correlated displays of the RFMDS and used to confirm shots/kills. The DIS time stamp in the PDU header only allows for an indication of the time passed since the current hour (23:86). This would cause some ambiguities on Red Flag missions which extend over several hours because events occurring during one hour of the exercise can not be distinguished between events happening in other hours. A zulu time stamp resolves all of these ambiguities and is therefore placed within the PDU.

Close examination of Figure 20 reveals that a significant amount of information about the aircraft flight parameters is available but not transferred to any of the fields in the DIS PDUs. The design of the DTC as it currently stands does not provide a means for this information to be transferred to the IVC. Future modifications to the DTC and IVC might develop a non-standard PDU specifically to transfer such data between components of the *RDT*.

3.3 Summary

Now that the CCC and the DTC designs have been presented, it is easy to see how these components can be referred to as the preprocessor components of the IVC. Blocks of data containing RFMDS messages are transferred from the RFMDS to the CCC using the V.35 protocol. The CCC parses the data block into individual RFMDS messages and transmits the messages, using the Ethernet protocol, to the DTC. The DTC parses each of the RFMDS messages and constructs DIS PDUs that are then broadcast onto a simulation network. The IVC described next, is located at one of the nodes on the simulation network and is capable of interpreting the DIS PDUs and rendering images which represent the activities described by the data.

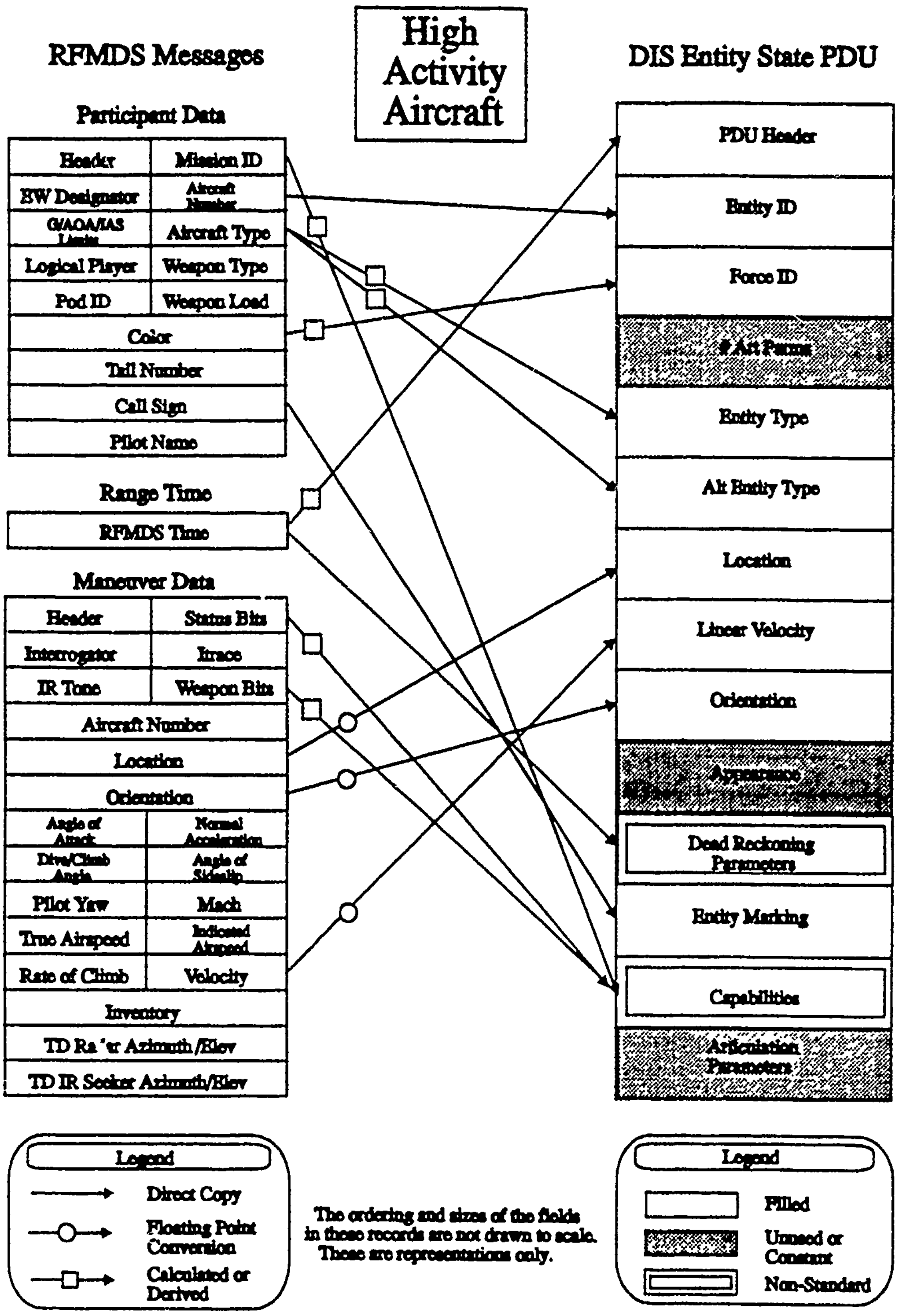

Figure 20. Entity State PDU Mappings for High Activity Aircraft

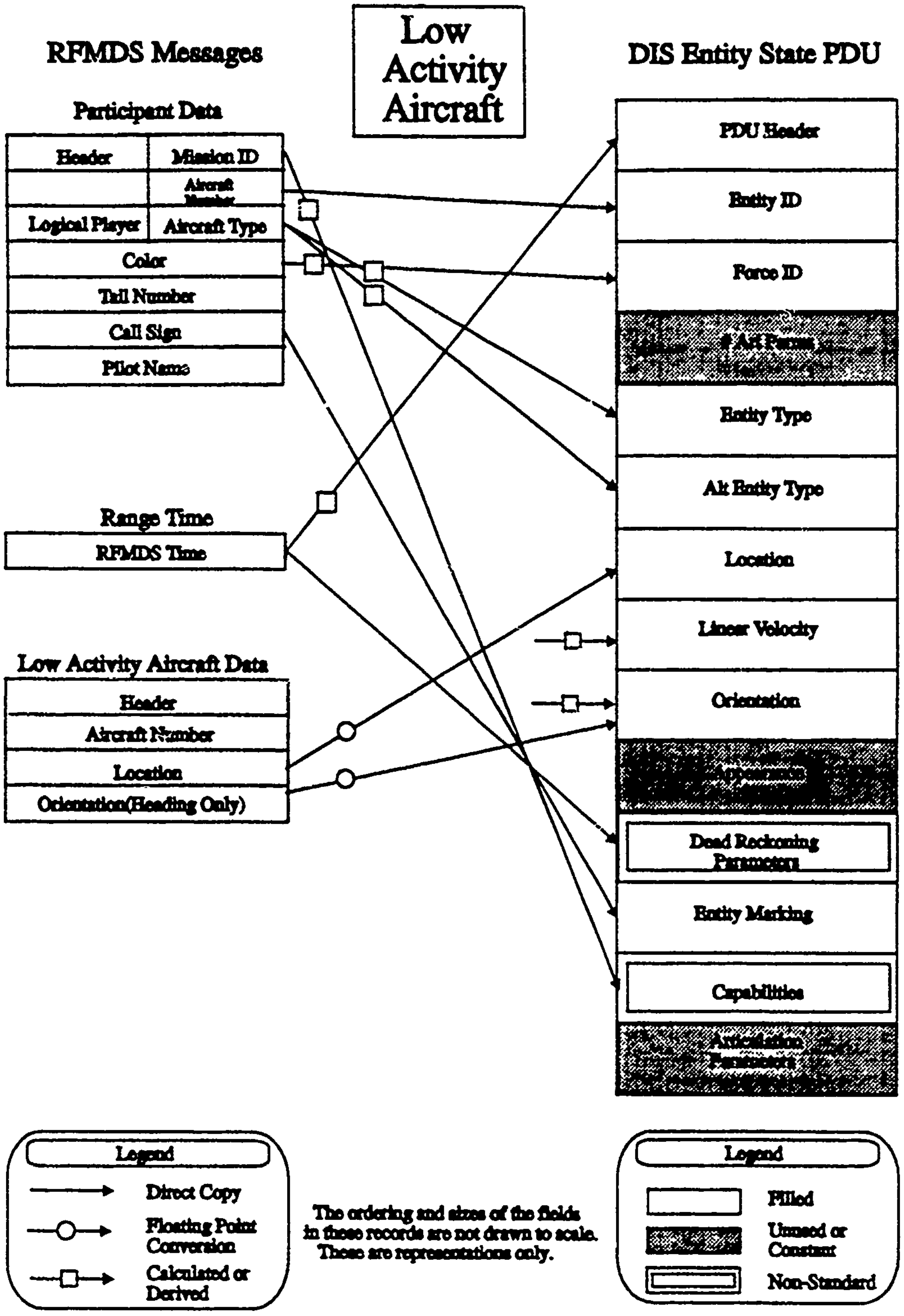

Figure 21. Entity State PDU Mappings for Low Activity Aircraft

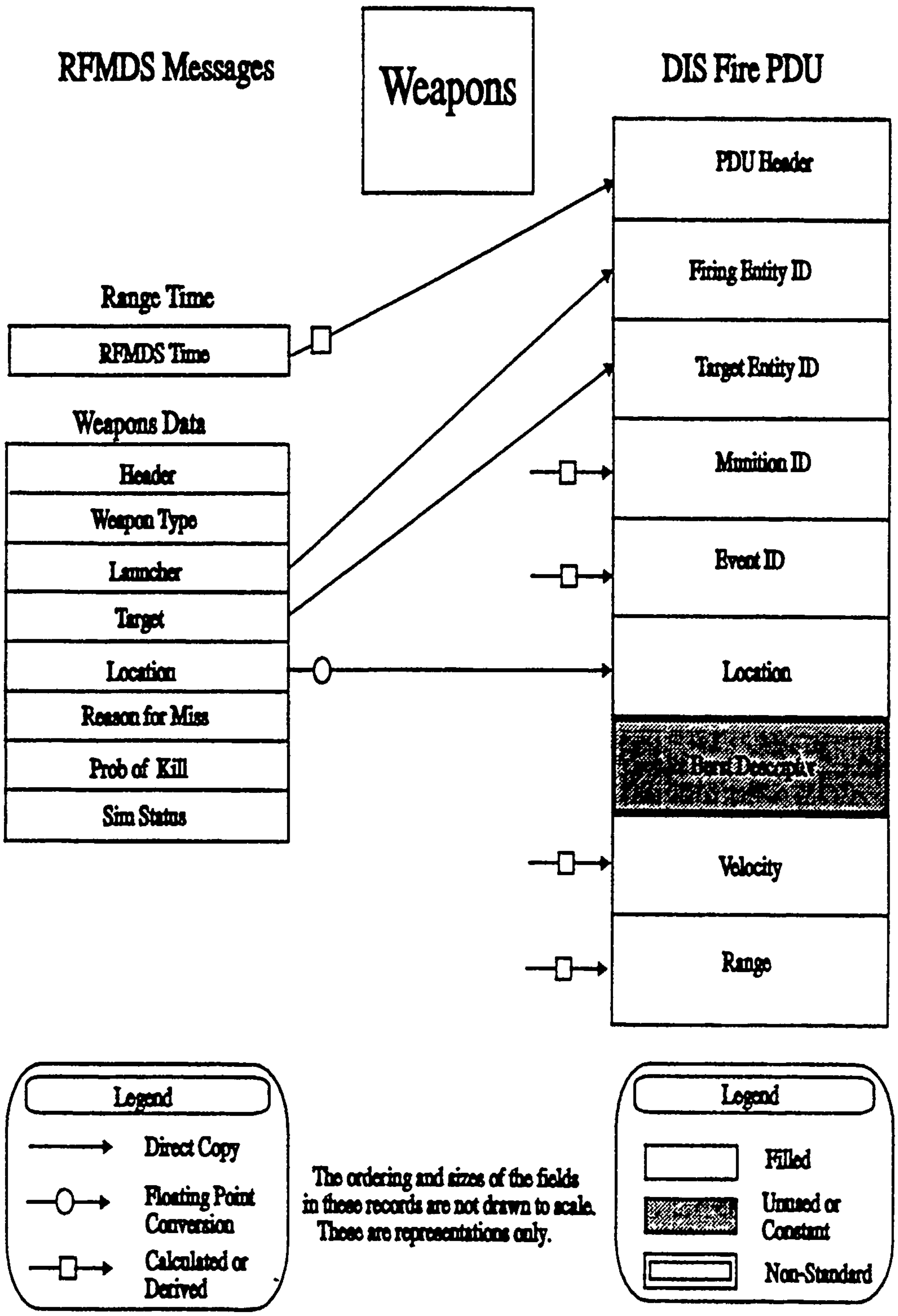

Figure 22. Fire PDU Mappings for Weapons

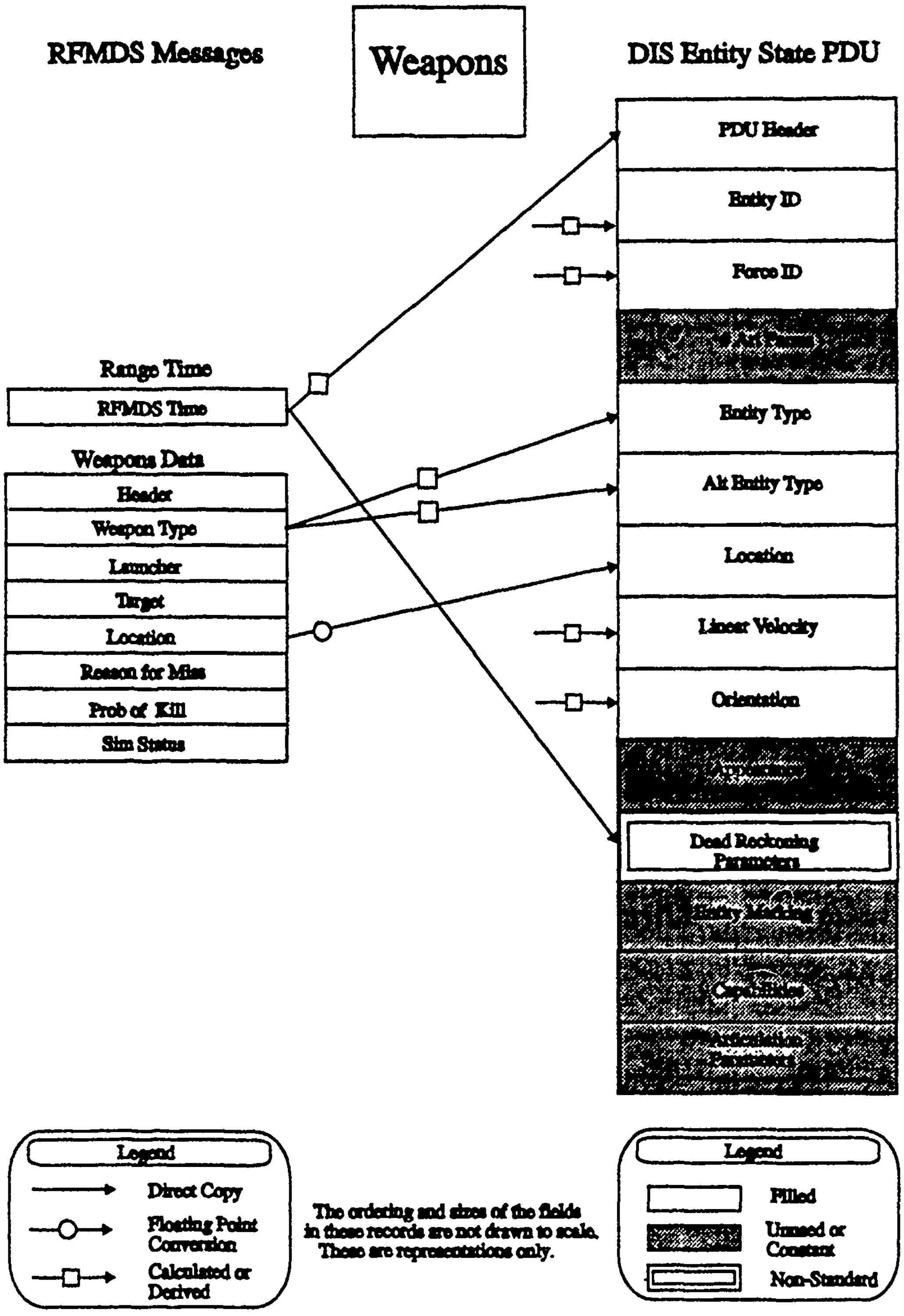

Figure 23. Entity State PDU Mappings for Weapons

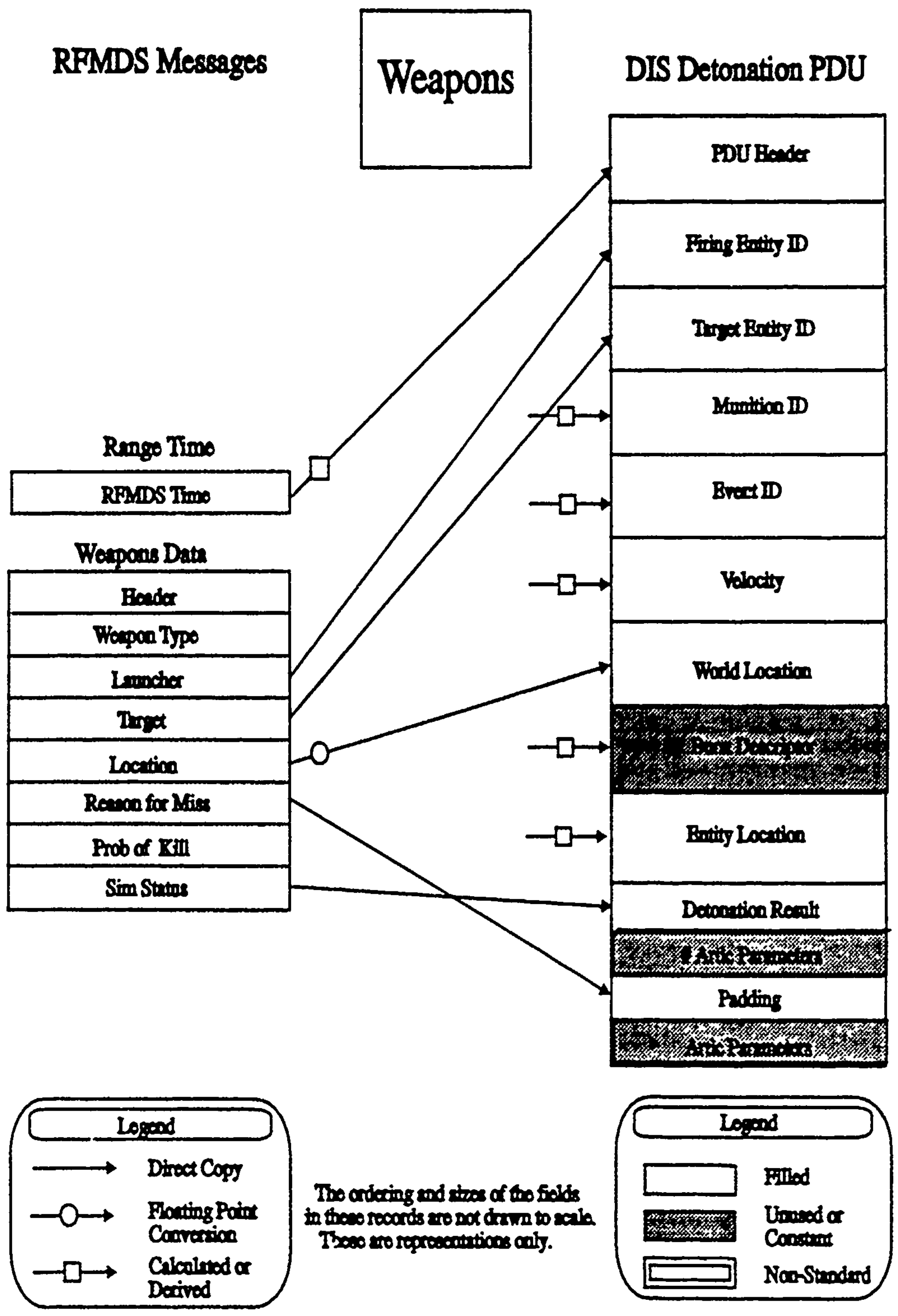

Figure 24. Detonation PDU Mappings for Weapons

IV. User Interface and Visualization Design

The organization of this chapter deviates somewhat from that of the previous chapter because of the numerous ways a user interface and visualization component (IVC) could be designed. It would be impractical, if not impossible, to identify and address every conceivable option that could be considered for the design of the IVC. Therefore, the approach taken in this chapter is to identify and describe the design methodology as it was used to create the interface portion of the IVC and then illustrate how the *ObjectSim* framework was adapted to meet *RDT* visualization needs.

4.1 User Interface Design

Foley et al. (5:391) define the key goals in user-interface design as:

1. Decrease time required to learn how to use the interface to perform a given set of tasks;
2. Decrease time required to perform the tasks with the interface;
3. Reduce the number of errors;
4. Encourage rapid recall of how to use the interface;
5. Increase attractiveness to potential users.

These goals form the foundation for the design of the IVC. A credible debriefing tool should be easy to learn and allow a user to quickly display crucial mission elements with minimal effort. A minimum effort implies, among other things, that little time is wasted correcting errors. Because opportunities to use the debriefing tool might be far apart, the procedure to perform tasks with the interface should be easily remembered.

In order to create an interface that achieves these goals, a methodology outlined by Foley (5:429) was used. This methodology encompasses the following steps.

1. Determine what the interface is meant to accomplish by learning what tasks are currently being performed and how they are completed.

2. Work through the conceptual, functional, sequencing and binding design levels to prepare a top-down design for the interface.
3. Use an interactive process of rapid prototyping and user testing to create a version of the interface with the desired functionality.

Each of Foley's design levels, conceptual, functional, sequence and binding, (5:394-395) are duplicated here for convenience. The *conceptual design* defines the principle application concepts that must be mastered by the user. The *functional design*, also called the semantic design, specifies functionality or meanings, but not the sequence of actions or the devices with which they are conducted. *Sequence design* defines the ordering of inputs and outputs and is also called the syntactic design. The *binding design* specifies how hardware devices/primitives are used to generate inputs and produce the desired outputs. The input primitives are the set of input devices that are available, including any of the following items: mouse, spaceball, keyboard, data glove, polhemus sensor, etc. Output primitives are the shapes, (such as lines and characters) and their attributes (color, font) provided by the graphics subroutine package. The presentation for this section is organized according to the three activities outlined in the methodology: interface definition, top-down design, and interactive prototyping.

4.1.1 Interface Definition. Mastering air-to-air combat tactics and maneuvers is one of the most challenging tasks facing aircrews in today's Air Force. The difficulty of the challenge is compounded by the very nature of air-to-air combat. No two missions are exactly the same and the lessons learned against adversaries on one mission may not be applicable to the adversaries, environmental conditions and scenarios encountered on other missions. Financial resources, once available, are dwindling and the high cost of generating sorties to learn and practice combat skills results in fewer sorties for individual aircrews. Each sortie represents an invaluable step towards achieving proficiency in the combat arena. It is said that "a picture is worth a thousand words" and in the air-to-air arena it is equally true for it is the image or set of images about a past engagement that are remembered long after the hours of discussion and verbal contention about what really happened during a mission have passed. Systems or tools which can graphically reconstruct the events of a sortie magnify the impact of any successes or failures because these graphic depictions can readily be cross-referenced mentally to the actual images seen during the flig' '. Thus one of the tasks of any debriefing tool must include an accurate portrayal of the mission environment from any point of view, especially that of the pilot.

Other tasks identified over the course of this thesis project are a result of discussions with analysts and test pilots from the 57TG as well as personal observations of mission debriefs that were conducted using the RFMDS. Enumerating these tasks will create a baseline functional description of the RFMDS capabilities that need to be mirrored in the final design of the IVC. The following list identifies many of the tasks that are currently being conducted at Red Flag. Each task description is accompanied with the method or procedure used by RFMDS to complete the task. (Note: This list is far from exhaustive in terms of the full capabilities of the RFMDS but does encompass the limited scope of the thesis project.)

1. Simultaneously present centroid, cockpit or plan views, and exercise/flight parameter data to the user. This is accomplished with three separate monitors at the DDS console.
2. Provide the capability to examine any area of interest from an overhead point of view by manipulating the plan view so that it can be enlarged/reduced and/or panned to any location on the range. Panning is accomplished by moving the joystick control in the desired direction. Twisting the zoom knob on top of the joystick enlarges the current scale by up to two times (22:4-12 – 4-16).
3. Focus the view area of interest upon a subset of high-activity aircraft and/or threats and allow the view orientation angles to be modified (Azimuth ± 180°and elevation 0 - 90°). Azimuth and elevation angles are modified via thumbwheels. Aircraft are selected by pressing the desired aircraft's select button (22:4-26 – 4-31).
4. Present a pilot view showing all high-activity aircraft and threats. Allow this view to be manipulated to show orientations from 12, 3, 6, and 9 o'clock. The view is selected via the PILOT VIEW button on the DDS console. A series of four buttons is used to select the desired view orientation (22:4-32 – 4-35).

5. Show a view from one of several ground-threat positions. This view is selected by pressing the THREAT BORESIGHT button and a threat select button (22:4-88).
6. Display the mission information about participants, mission role, call sign. Pressing the EXER DATA button on the console selects one of several screens containing information about high-activity, low-activity and threat participants (22:4-40 – 4-44).
7. Show dynamic flight data for each aircraft to include mach number, true airspeed, indicated airspeed, angle of attack, pitch, roll, heading, etc. (22:4-44 –4-45) Select this view by pressing the PILOT DATA button.
8. Provide system/flight parameter display useful for technical evaluation. Some of these parameters include normal acceleration, mach number, angle of attack, position components, velocity components, tracking filter status and others. This view is selected by pressing the ENGR DATA button (22:4-45 – 4-47).
9. Present a chronological list of significant mission events to include simulated weapon firings and detonations along with the results. The list should also include the bearing, range and closing velocity from the weapon to the target. Scan vertically through this list of events by using scroll bar buttons on the DDS console (22:4-52 – 4-53).
10. Allow aircraft-aircraft pairing. This provides the display of slant range, bearing, closing velocity, altitude difference, angle off the tail (aspect angle), and antenna train angle. The display should be capable of pairing up to eight aircraft and is selected by pressing the FLIGHT DATA, A/C A/C, AND ROW/COL buttons (22:4-70).
11. Show infrared (IR) and radar missile-lock indications (22:4-72). Software controlled.
12. Display representations of the radar and IR missile seeker angles on the pilot view with square and diamond icons. This symbology is available by selecting ACM mode when in the pilot view (22:4-76).
13. Monitor aircraft communications on up to eight channels. Enable the desired audio channel via buttons and volume control knobs on the DDS console (22:4-114).
14. Depict a history of the aircraft's flight path by showing flight path trails on the views. This is selected via the HISTORY TRAIL button on the console.

The following additional tasks were also identified as desirable but are not part of the RFMDS capabilities.

1. Provide a view into the scene from a position aft of an aircraft along its velocity vector. This corresponds to a wingman's view from a trail position. Allow manipulation of the view orientation in any direction.
2. Show an aircraft's bearing and range from a known geographic reference point. This point, known as a bullseye point, is used by weapons controllers as a standard ground reference for describing aircraft locations to multiple dispersed flights.
3. Detach from a wingman's view and "fly" through the scene to achieve a view from any conceivable position or "chase" mission participants through their maneuvers as a "phantom" wingman.

These lists represent a significant subset of the tasks necessary to completely analyze an air-to-air exercise and present a cohesive picture of all of the aircraft activities during a mission.

4.1.2 Top-Down Design. The designs necessary to achieve the capabilities described above include conceptual, functional, sequence, and binding. The following subsections detail the composition of each of these levels.

4.1.2.1 Conceptual Design. The initial design strategy of the *RDT* was *not* to create a totally new system requiring extensive periods of retraining in the way a debriefing might be analyzed with a new tool, but rather to mimic the familiar ways of the RFMDS as much as possible and then make new methods available which can extend current capabilities. Achieving this goal requires the consideration of three main factors.

1. The *RDT* is to be implemented on a workstation with a single monitor unlike the RFMDS DDS console with its three monitors that can simultaneously present three different views into the mission. Thus monitor "real estate", or the amount of screen space available for the various views, is the driving factor in the overall design for the IVC. The need to display simultaneous views suggests the use of a windowing

technology and a window manager to manage the sizing and placement of views on the workstation. Multiple views displayed simultaneously on the workstation implies that the size of the individual windows must be small enough so that one window does not cover up the contents of another. Small windows, however, obscure details that might otherwise be visible in a larger window and thus suggest an additional feature that would allow a window to be expanded up to the full screen size of the monitor.

2. Custom hardware control devices such as the thumbwheels, joysticks, and large button arrays of the DDS console work well with the DDS but are not readily available and would require a significant effort to manufacture and interface into the SGI computer hardware and software. New controls for manipulating the views and data should be simple to use, without sacrificing controllability; and they should not require an inordinant amount of time to make the transition from one system to another. These factors suggest the use of common computer interface devices, such as the mouse or keyboard, to provide the needed viewing control functions and operating system interface.

3. The IVC viewing controls should allow the tasks listed in the preceding subsection to be completed at least as easily and quickly as the controls of the RFMDS and still provide a measure of extendibility. (Easy, in this context, is defined as requiring minimum numbers of controls to perform a single task, as well as a minimum amount of memorization as to the location and function of each button, knob, dial, etc.) This criterion suggests the need to keep a simple set of powerful controls contained within a small area so that time is not wasted due to large hand movements which might be necessary to control either the devices themselves or the views. These controls should also provide a consistent appearance and response so that errors are minimized and the application of one control device is easily transferrable to other tasks.

Figure 25 represents the *RDT* conceptual interface design derived from the major considerations identified above and from the list of tasks specified earlier.

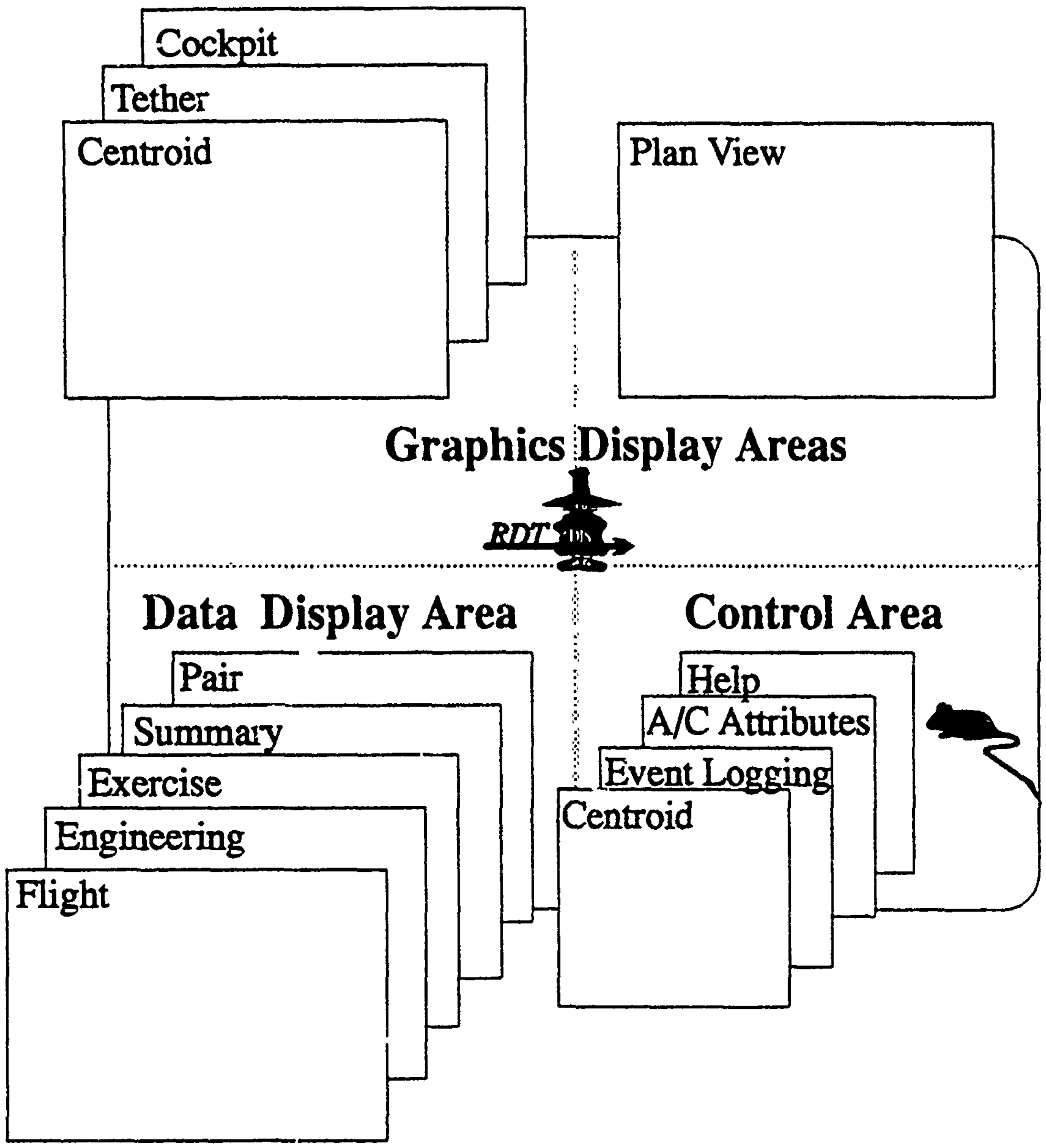

Figure 25. *RDT* Conceptual Design

The monitor screen space is partitioned into four equal areas or windows. The top t·. areas are set aside for the plan, or overhead, view and one of three possible 3D-views. The 3D views are the centroid view, the cockpit or pilot view, and the tether view. The centroid view focuses the viewing volume upon up to four aircraft and automatically repositions the view volume to keep these centroid aircraft in sight at all times. The tether view, or the wingman's view, shows a display from a trail position behind any of the aircraft as they maneuver through the environment. The cockpit view is self-explanatory.

The lower left-hand area is reserved for displays requiring the presentation of textual data and is referred to as the data display area. The data displays are the flight data, engineering data, exercise data and summary data. The flight data display shows the values of some of the aircraft flight parameters available to a pilot. The engineering display provides technical information regarding aircraft position in the RFMDS coordinate reference frame. The exercise data presents a list of all of the aircraft which have been active on the range, their identities, and their roles. The summary data display provides a chronological list of all of the weapon events received during the mission.

The lower right-hand area is used exclusively to control each of the views and data displays through a mouse driven graphical user interface (GUI). This area is referred to as the control area. Use of the mouse and GUIs to directly manipulate control objects minimizes the need to memorize numerous keyboard sequences and provides familiar visual representations which can easily be assimilated. Some objects on the control panel also create other mini-control panels that are overlaid onto the data display area and removed when no longer needed.

Each of the graphics windows (3D views or plan view) in the top areas can be expanded to fill the full width of the monitor or enlarged to fill the full screen. While a view is enlarged to fill the full screen, the control panel window is hidden. Manipulation of the views is restored by bringing the control panel back into view and operating the control objects. The mini-control panels available are the help panel, centroid panel, attributes panel and the event logging panel. The help panel provides information regarding the manipulation of control panel objects. The centroid panel specifies which aircraft are to be centered within the centroid view. The attributes panel allows aircraft attributes, such as color and length of flight path history trails, to be modified while the event logging panel identifies the filename to be used as an archive for the weapon events.

The combination of the plan and 3D views, data displays, and the control panels comprise the *RDT* user interface and work in concert to achieve each of the tasks identified earlier. As with all things, there are a few exceptions to this statement. Audio communications cannot be monitored by the *RDT* at this time. This capability was identified as outside the scope of this thesis. Display representations of the radar and IR seeker angles have likewise been postponed pending resolution of design and implementation issues. The procedures or sequences of steps needed to perform each of the listed tasks within the framework of the *RDT* user interface design are identified in the next section as part of the functional/sequence design of the *RDT*.

4.1.2.2 Functional and Sequence Design. In keeping with the design philosophy stated in the previous section, mimicking old ways, the functions of each of the controls of the DDS console were used as a template for developing the user interface. Each knob, button, dial, and joystick of the DDS console was examined and categorized according to its function and context in which it was used. GUI objects were selected which emulated the functionality of the DDS controls. This does not mean, however, that the GUI objects are identical in shape, size or location to the DDS controls. As an example, a thumbwheel on the DDS console is used to rotate the centroid view about a vertical axis; a slider designed into the *RDT* performs an identical function although the appearance is somewhat different.

The functional design specifies the meanings behind a series of actions; whereas the sequence design defines the ordering of the actions. This section strays from a pure portrayal of either design and instead combines the action sequences into a series of state transition diagrams reflecting control panel object manipulations. In nearly all cases, the user interface merely sets/resets information that the application uses to control the visualization. Separate functional definition diagrams would be of little value and are not provided. The format used to illustrate the diagrams generally follows the format defined by Rumbaugh (17:84-112) for his dynamic model notation.

One concept not readily apparent by looking at the state transition diagrams is the ability of the IVC to exist in multiple states concurrently. This property is called shared control and is used in some user interface management systems in which subroutines are executed in response to user inputs. (This concept is described more precisely later in the discussion about rapid prototyping.) As long as a GUI object is capable of "recognizing" an input, its subroutine will be called. In the *RDT* design, any sequence of buttons may be pushed as long as a button remains "active." An active button, or GUI object, responds to manipulation and causes a state transition. An inactive GUI object cannot respond to manipulation and therefore cannot cause transitions to intermediate states. As an example of this shared control, a user can change the current view type while he is in the process of modifying the flight data control information. By activating/deactivating buttons, a path through a set of intermediate states can be set up which completes a complex activity. Any beginning state which establishes a complex activity is called a "complex state," or "command state." The IVC can be transitioning through several states at the same time as long as buttons remain active on the control panel. A line with arrowheads on both ends signifies a transition to a "command state" and a concurrent return to the "neutral state." The dynamic model for the entire interface portion of the IVC is shown in Figure 26. The initialization state is not shown. Shaded boxes indicate "command states" which are diagrammed in figures contained in Appendix A.

4.1.2.3 Binding Design. The binding design determines how input and output units of meaning are formed from hardware primitives (5:395). In the case of the *RDT* this translates to the selection of specific input devices and to the output primitives necessary to create the scenes displayed on the monitor. The primary focus of this subsection is on the input devices chosen for the *RDT*. The output primitives, namely fonts, lines, line width, polygons, colors, and textures are largely determined by the geometry of the models used to represent the terrain and aircraft flying in a Red Flag exercise. The *Performer* software described earlier is responsible for rendering the geometry, therefore no further hardware output binding design is given.

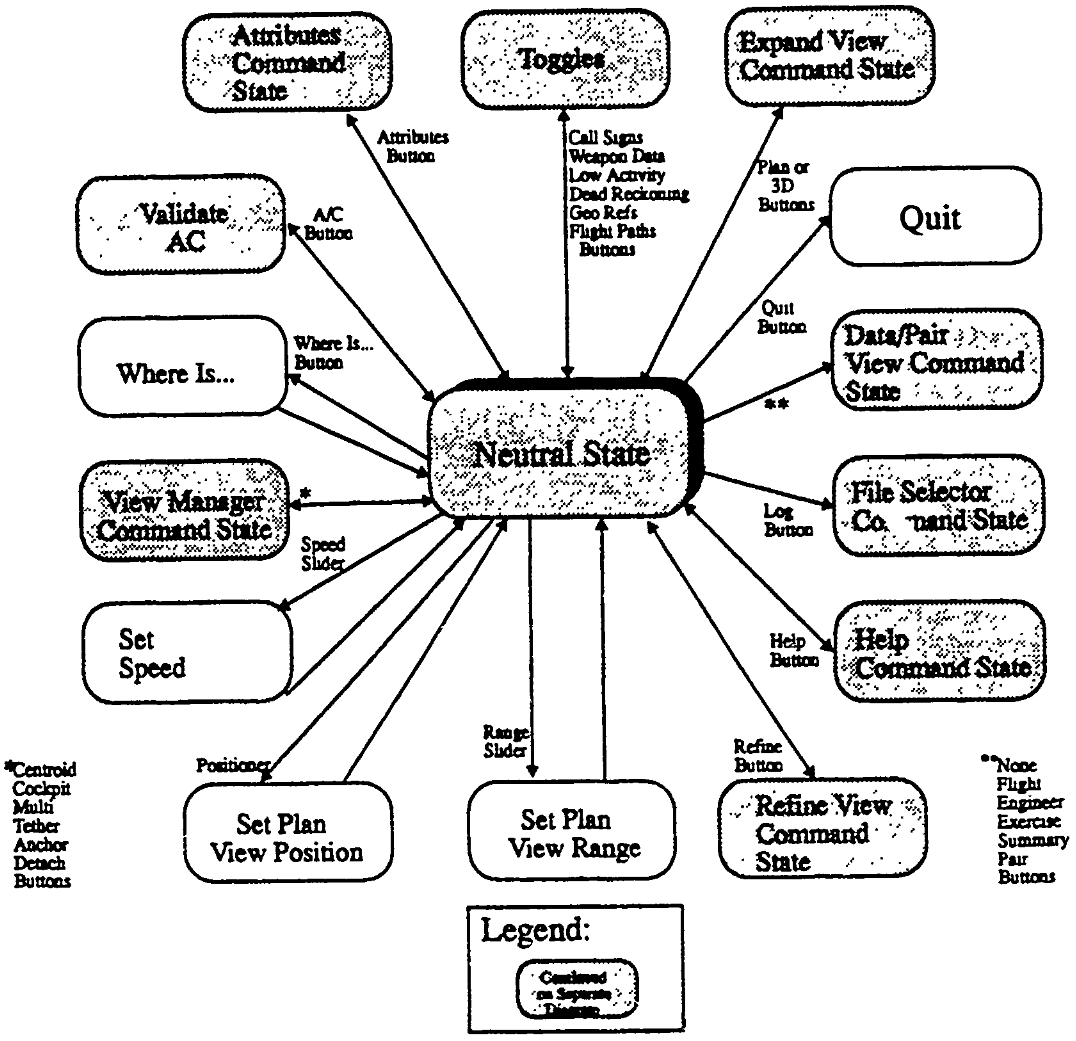

Figure 26. IVC Dynamic Model

A number of input devices were considered for incorporation into the *RDT* but only three were selected for the first implementation. The following list identifies the devices which were considered and the reason that each was either adopted or disregarded. (While reviewing the list it is important to remember that the purpose of the *RDT* is to prove the feasibility of implementing a tool for remote debriefing that uses the DIS protocol and not to specifically conduct research regarding the optimum set of input devices. The task is to create a set of suitable devices that make control of the visualization consistent, quick, easy, and error free.)

1. *Keyboard* – This device is included with every workstation and forms the backbone for communicating with the processors. Using any of the keyboard keys to control events within the interface allows rapid switching of control parameters. Familiarity and availability of the keyboard make the keyboard a must for *RDT*.

2. *Mouse* – This input device is included as part of the *RDT* for basically the same reasons as listed for the keyboard. A mouse is included with each workstation and is a familiar input device. It is one of several direct manipulation devices that permits smooth hand motion to translate into smooth cursor positioning. Precise positioning is more natural and easier to accomplish on direct manipulation devices than on discrete devices, such as keyboards (5:351). In the world of "windows" and "window managers" a mouse lends itself well to the manipulation of GUI objects because of the preponderance of software written which utilizes the mouse for accomplishing selection and positioning activities.

3. *Voice Recognition System* – A voice recognition system would provide an excellent hands-off view manipulation device; however, work with a voice recognition system at AFIT confirms the experience of others regarding voice recognizers, namely that there are significant limitations. Some of these limitations include: the need to recalibrate the recognizer for each new user (variations in the voice wave patterns, such as occurs with a cold, also require recalibration), a limited vocabulary, and the requirement to pause between words to signal that the end of a word has occurred (5:355).

Delays encountered recalibrating the device for each new user could make the overall system seem unattractive in an environment where new users change frequently and time management is critical. Adoption of a voice recognizer into the *RDT* must wait until these limitations can be overcome.

4. *Joystick* – The joystick is another direct manipulation device which could be used as part of the interface. They are readily available and easy to use; however, they are generally limited to movement with only two degrees of freedom and are awkward if movements other than pitch and roll are desired (i.e. a twist.)

5. *Spaceball* – A third direct manipulation device considered for the *RDT* is the spaceball. Like the joystick, precise positioning is difficult yet the design of the spaceball often includes an arm support which allows the fine motor control muscles of the hands and wrists to be used. Unlike the joystick, the spaceball allows movement over six degrees of freedom and greatly enhances the ability to control view orientation angles in any direction.

6. *Other* – Other high technology devices such as the data glove and Polhemus tracker were considered impractical because of time delays involved with calibrating and donning/doffing the devices, the level of effort required to implement the interface, and the scope of the thesis project. Future modifications to the *RDT* might reconsider using alternate input devices.

The final design of the IVC calls for a keyboard, mouse and spaceball as the standard set of input devices.

4.1.3 Interactive Prototyping. The final step in the design methodology outlined in this chapter dictates that an interactive process of rapid prototyping and user testing be used to narrow down the final interface design. This subsection highlights the characteristics of a user-interface management system (UIMS) that was used to develop the initial prototype, and presents the final designs chosen for the GUI objects as a result of iterative design techniques.

4.1.3.1 User-Interface Management System - Forms. The development of the initial prototype was created by using a UIMS called *Forms.* A UIMS is a software tool which can assist in defining not only the form of an interface, but also admissible action sequences. Some UIMSs also provide an interactive design medium through which all of the attributes of a GUI can be defined, for example size, color, location, and input response. Another characteristic of a UIMS that makes it attractive as a tool for developing interface prototypes is the concept of shared control. An application built on top of a UIMS is typically written as a set of subroutines, called semantic action routines, which are called in response to user inputs. The UIMS is responsible for calling the appropriate semantic action subroutine to complete the desired task. (This is the "call-back" paradigm mentioned earlier.) In return, these action routines influence the set of acceptable action sequences available within the application and dialog control is shared between the application and the UIMS(5:457).

The *Forms* UIMS was developed as a library of subroutines that can be used to build up interactive forms of buttons, sliders, input fields, dials, etc. in a simple way for Silicon Graphics workstations. It was written by Mark H. Overmars at Utrecht University, the Netherlands, to overcome the problems of high cost, limited capabilities, and difficulty in using other UIMSs, and Forms is available in the public domain. His design goal was to create a tool that was simple to use, powerful, graphically good looking, and easily extendable (13:i). The *Forms* UIMS uses the shared control concept described previously and relieves a programmer of the burden of mapping device inputs to appropriate action routines.

The *Forms* library also includes a design tool which facilitates the construction of forms by interactively allowing GUI objects to be placed, scaled, and moved in a simple way. Object attributes, such as color, labels and fonts, can be changed easily (13:35). This tool, called the *Form Designer*, was the primary development tool used to construct the interface prototype of the IVC. The design process used to create the interface prototype proceeded in the following manner.

1. A color scheme was chosen to provide a pleasing appearance and consistent mapping between colors and activities associated with them. Because color interactions can be a complex issue, a known, commercially proven, color scheme was chosen to minimize development time and benefit from previous research. The *RDT* reproduces the colors of the *Turner* color scheme used by Silicon Graphics in the *Case Vision* software.

2. The *Form Designer* was used to create the various interface forms dictated by the conceptual design and containing all of the required GUI objects. These objects were grouped according to functional tasks, and action sequences were defined in terms of "call back" subroutines. Forms "call back" subroutines are procedures that are invoked in response to a Forms event, like pushing one of the buttons or moving one of the sliders.

3. "Call back" subroutines were written and then integrated with the *Forms* subroutines into a prototype interface.

4. The interface was tested by personnel of the 57TG and by former RFMDS console operators to ensure consistency, speed of use, and correct sequence of actions. Errors which were discovered were subjected to a subsequent iteration through the *Form Designer* and/or revision of the "call back" routines as required.

Although this design process began well before any of the visualization software was implemented, it continued throughout the entire thesis project. As new methods and features were added to the IVC, successive repetitions through the prototyping process were conducted until a suitable interface was developed which met the overall interface design goals identified at the beginning of this chapter.

4.1.3.2 Final Interface Design. The following sets of figures contain the final interface forms created with the *Forms Designer.* They are reproduced here to show the format and content of the final design. The color scheme can be observed by referencing Appendix E. The illustrations are presented according to functional groupings with the master control panel pictured first in Figure 27. Each of the *Forms Designer* created forms, now called panels or windows, corresponds to tasks which are available from either the "neutral state" or one of the other "command states." The master control panel

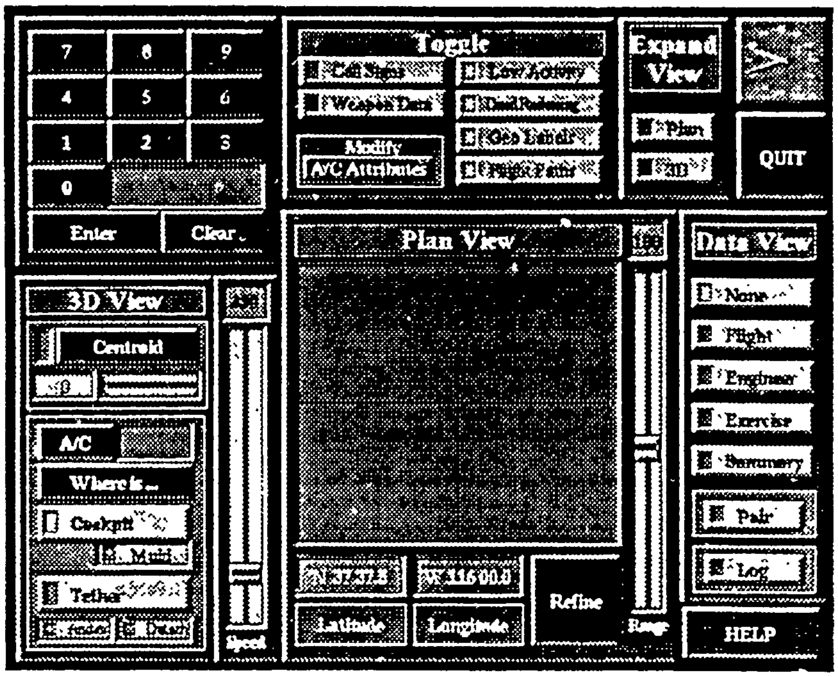

Figure 27. Master Control Panel

corresponds to activities available from the "neutral state." The centroid panel in Figure 28 corresponds to the view management command state. Tasks available in the data view/pair command states are activated via GUI manipulations on the data view or pair panels depicted in Figures 29, 30, 31, 32, and 33. The help panel shown in Figure 34 coincides with the help command state. The panel in Figure 35 is used in the refine command state while the panel in Figure 36 is used in the attributes command state. The last panel presented in the series is actually the first panel to be displayed upon IVC start up. Figure 37 is used to set the initial view and view modifier configurations of the IVC. The following section describes how the user interface is incorporated into the *ObjectSim* framework to form the complete IVC.

4.2 Visualization

The visualization software contained within the *RDT* is principally built around the *ObjectSim* framework developed by Snyder (20). Aircraft and terrain geometry models are rendered by SGI hardware and *Performer* software through the *ObjectSim* interface. The

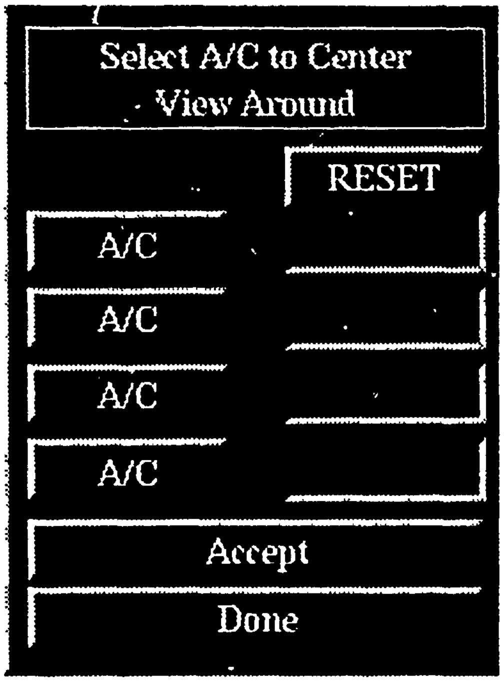

Figure 28. Centroid View Control Panel

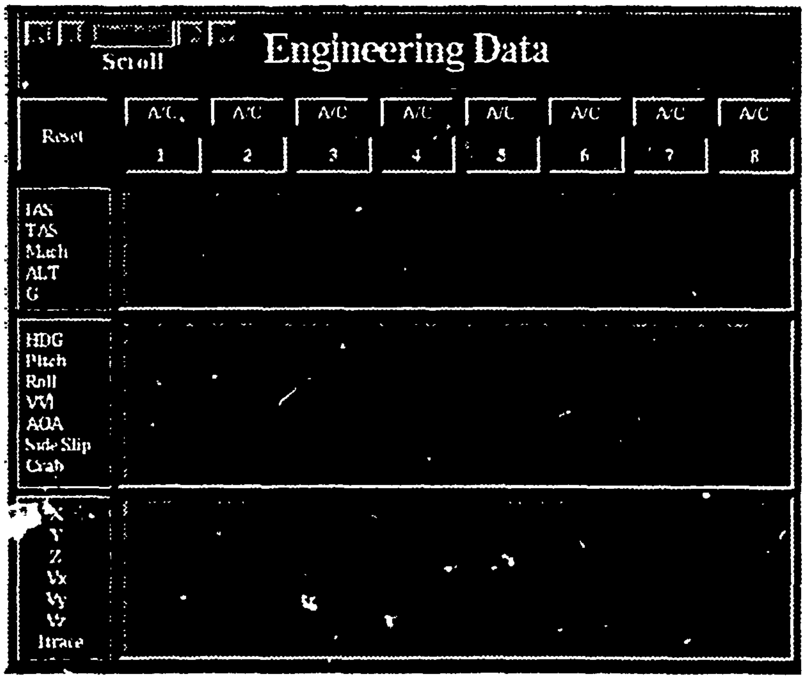

Figure 29. Engineering Data View Panel

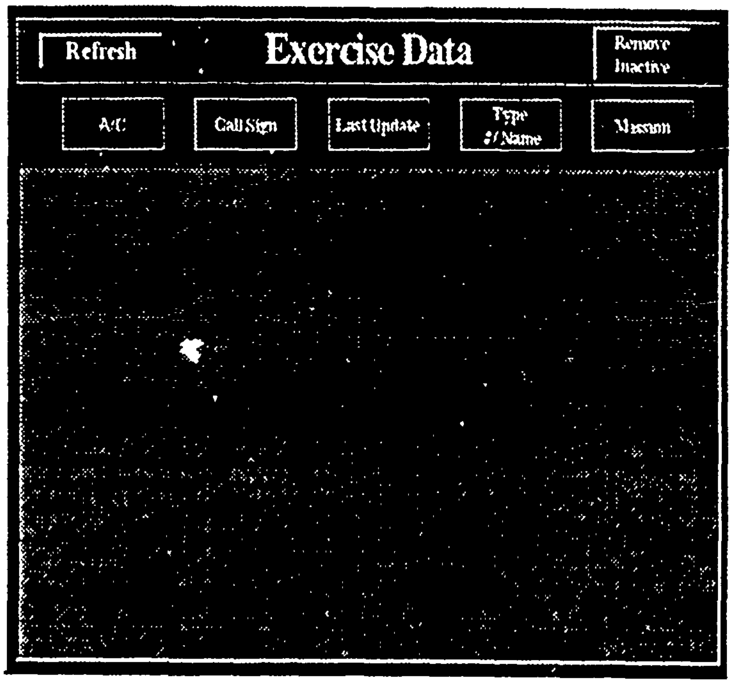

Figure 30. Exercise Data View Panel

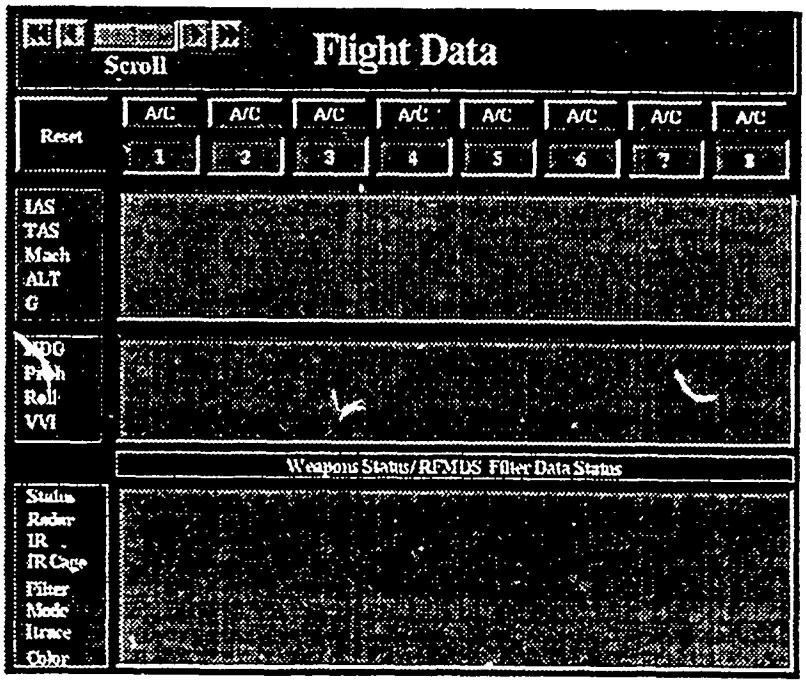

Figure 31. Flight Data View Panel

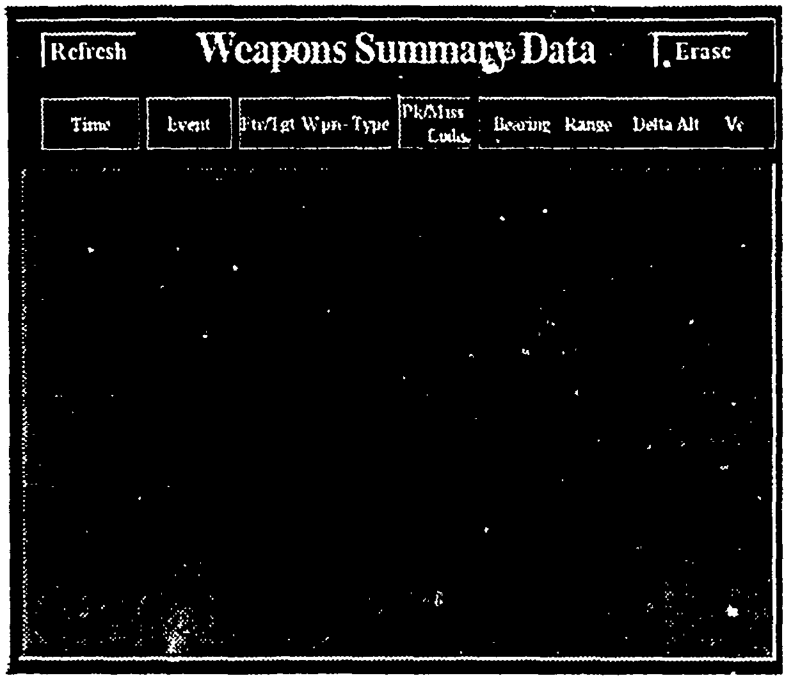

Figure 32. Summary Data View Panel

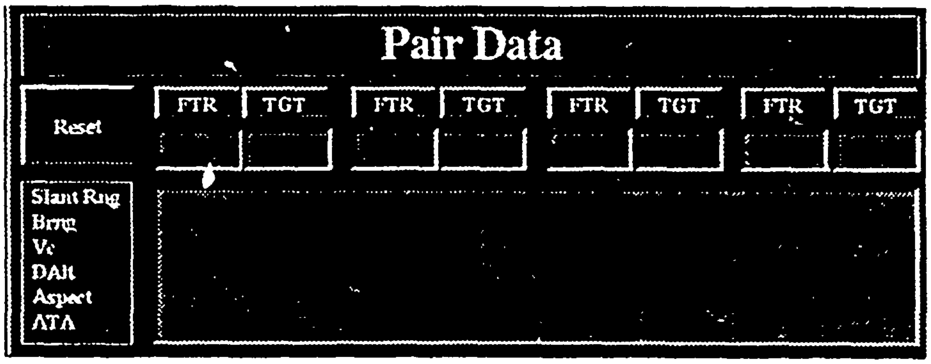

Figure 33. Pair Data View Panel

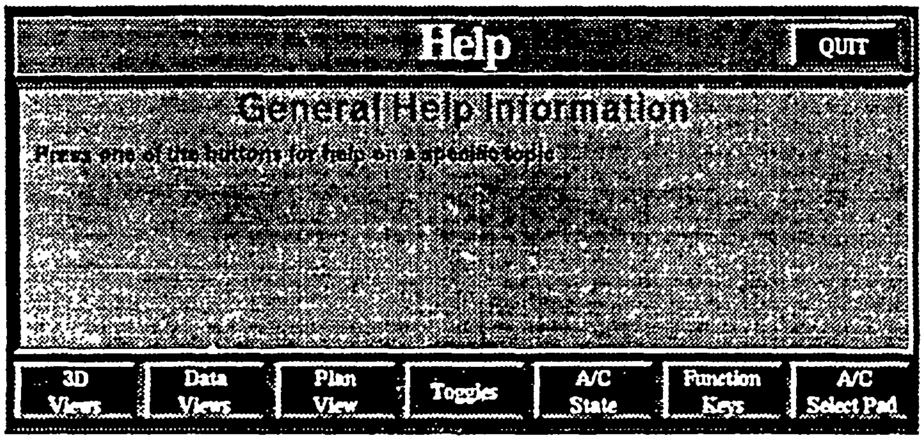

Figure 34. Help Panel

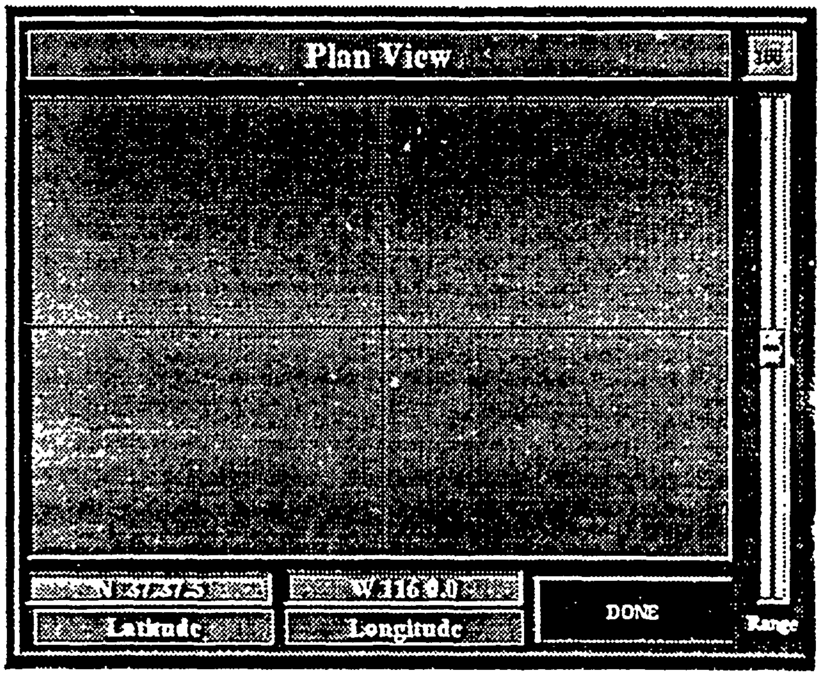

Figure 35. Refine Control Panel

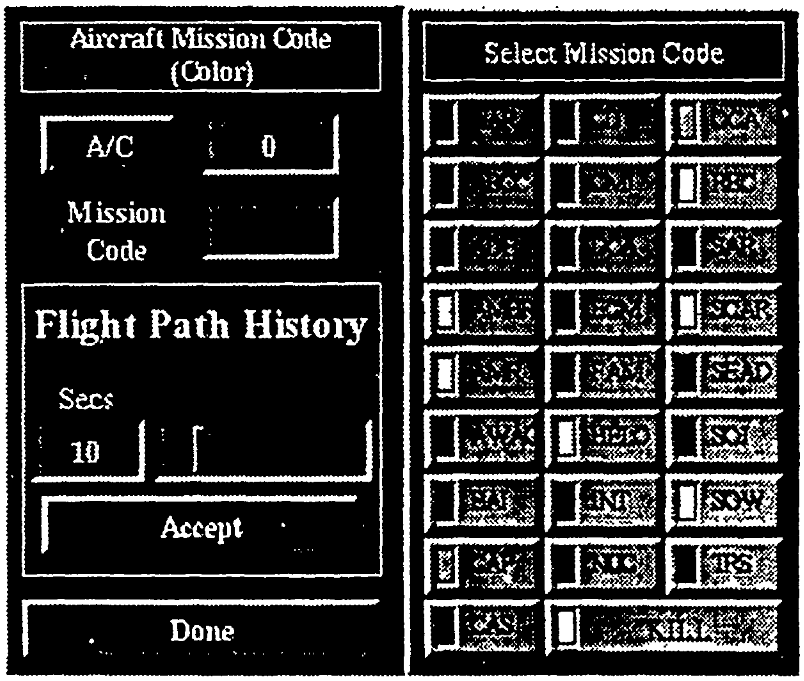

Figure 36. Attribute Panels

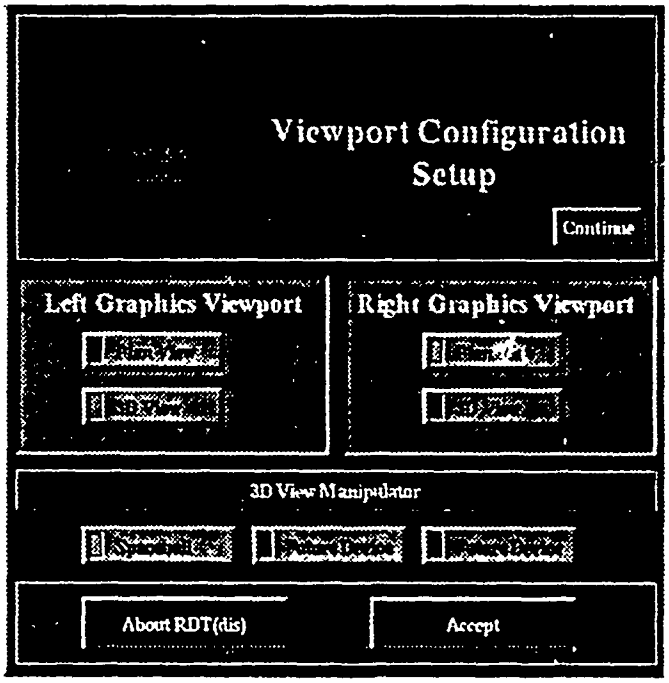

Figure 37. Configure Panel

design of the IVC uses the principles of inheritance to derive subclasses of the *ObjectSim* class structure. These modifications to the basic framework allow the *RDT* application to be tailored to the specific needs of the Red Flag environment. This section briefly describes the *ObjectSim* class structure and then shows how *RDT* specific classes are incorporated into the overall design of *ObjectSim*. The section concludes by describing the design considerations needed to integrate the *RDT* user-interface with this *ObjectSim* application.

4.2.1 ObjectSim *Framework.* *ObjectSim* was designed as a set of reusable components in a C++ class library to provide a high-level wrapper around the *Performer* library and its programming paradigm, namely a wide interface into an image generation data structure. It is intended to be a standard interface that encapsulates the common functionalities of DIS simulations into readily-available, high-level services. Visual simulations, in their simplest form, consist of a basic set of classes: application, model, and renderer. The application propagates one or more dynamic models through the scene while the renderer displays the corresponding geometry depicting the simulation. The *ObjectSim* framework builds upon these simple concepts and creates a set of classes which not only perform these simplistic activities, but also provide built-in functionality for managing terrain, attaching views, and interfacing with various input devices such as helmet mounted displays or spaceballs.

Figure 38 is a high-level representation of the *ObjectSim* framework. Rumbaugh object model notation (17:21- 57) is used to convey the relationships between classes. The Simulation class contains zero to many Player objects, a Renderer object, a Terrain object, multiple View objects and zero-to-many Flt_Model objects. The Simulation class is responsible for multi-player tasks such as propagating players, switching views, and attaching views to players. The Player class is used as a repository of state and attribute information about specific entities participating in a simulation. In addition, the instances of the Player class, known as View players, may also be associated with a view and perform their own draw functions using SGI graphics language (GL) subroutines. The Terrain class encapsulates the movement of the terrain into a near-transparent operation so that either round or flat earth operations can be performed with minimal effort. (In *ObjectSim*

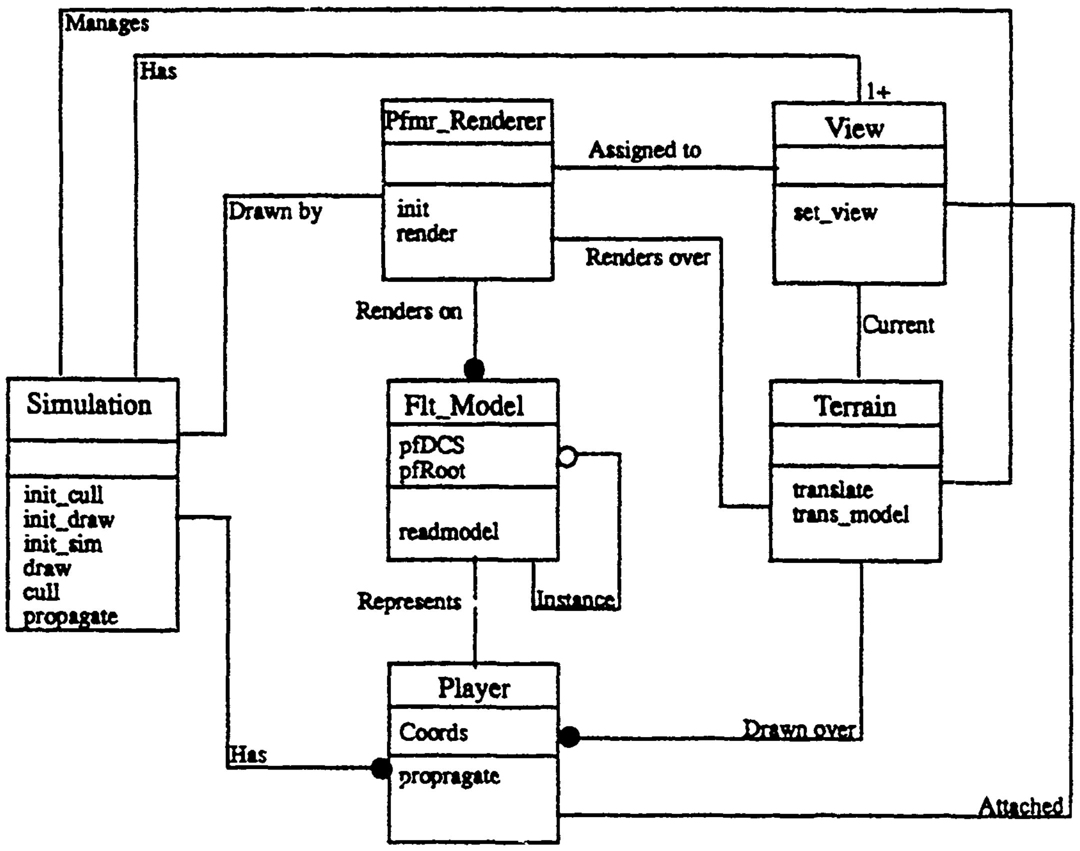

Figure 38. *ObjectSim* Object Diagram

terrain is moved under a view player instead of moving the view player over fixed terrain. This is done to minimize floating point precision inaccuracies which occur as a result of player movement located at some far distance from the terrain origin. Keeping the view player fixed at the origin and moving the terrain relative to the player ensures sufficient precision to express the exact location and guarantees that no visual artifacts, such as jitter, appear.) A View object is associated with the Renderer and images drawn by Renderer are assigned to a View. View modifiers dynamically allow devices such as spaceballs or helmet mounted displays (HMDs) to change the view orientation parameters for total control of the viewpoint and view orientation. The Flt_Model class represents the geometry of active entities in the simulation.

4.2.2 RDT *Modifications.* Tailoring of the *ObjectSim* framework is accomplished by deriving subclasses from the basic class structure and overriding methods to perform new tasks. The IVC design for the *RDT* requires modification of the Simulation, Player, and View classes. Figure 39 shows the *RDT* derived classes along with a set of new classes needed to complete the full design of the IVC.

The RDT Application is a Simulation which has been modified to perform additional initializations specific to the RFMDS environment such as loading translation tables which map RFMDS aircraft types into DIS entity types. The RDT View is a View with additional methods that allow it to attach the rendering to one of the view players. The *ObjectSim* Player is used as the base class for two new RDT players, the RDT Player and the RDT View Player. The RDT View Player has additional methods which perform GL drawing into the scene. The RDT Player contains attribute information that is not part of the base Player class. Using the methods in the RDT View Player, the identity of the active RDT Players can be shown in the form of a number or their specific call sign.

A set of new classes is needed to interface the *ObjectSim* framework to the communications network where the DIS PDU information is being received and interpreted. The Object Manager, developed by Sheasby (18), is the bridge to the distributed simulation network. Methods accessible to the IVC allow network information to be incorporated into the simulation. The RDT Net Manager is responsible for polling the Object Manager for

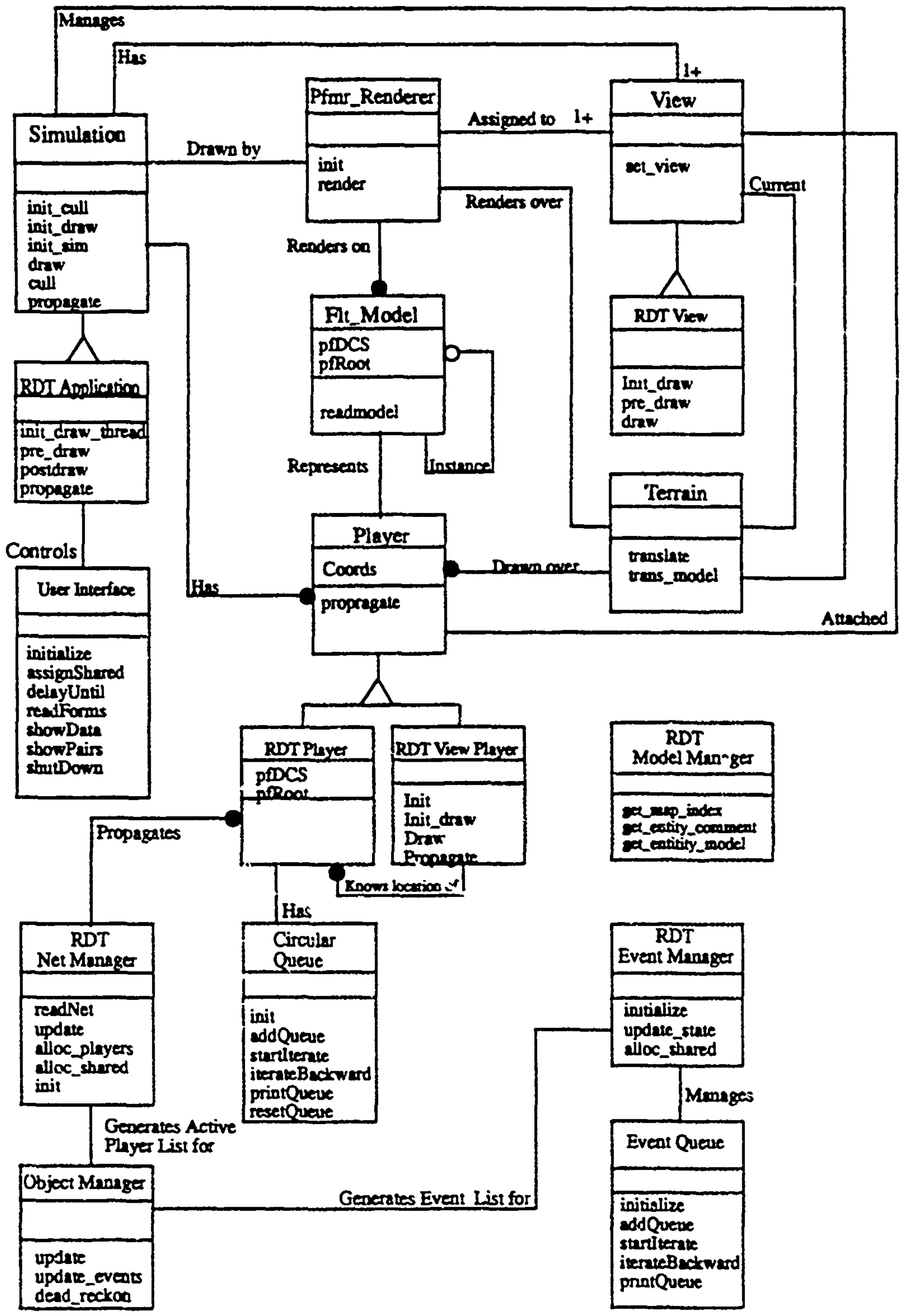

Figure 39. *RDT* Modifications to *ObjectSim*

current network information and then propagating each of the RDT Players through the scene based upon either new or dead reckoned coordinates. The RDT Event Manager also polls the Object Manager and accepts a list of current events which have been received since the last update. The events are stored semi-permanently in an Event Queue and are used to trigger animation sequences in the scene such as weapon detonations. The RDT Model Manager contains methods to determine the location of the appropriate geometry files that are needed to associate with an RDT Player.

4.2.3 User Interface Integration. The integration of the user interface with the RDT Application is designed such that all interaction with the RDT Application is done through a common control data structure. Changes to desired view parameters are stored in the shared structure by the user interface. These changed parameters are then read and interpreted by the Application and used to adjust the simulation views as requested. This shared control structure design makes it possible to completely change the form of the user interface without affecting the code or operation of the RDT Application. In reality, the user interface is more than just a single class. It encompasses all of the *Forms* semantic action subroutines into a Controller class as well as general utilities that are needed as part of the initialization of the interface. Because the *Forms* software is generated in the "C" language, the user interface is also the bridge between the "C++" RDT Application software and these "C" subroutines. Figure 40 illustrates the communication between the two activities or processes through the shared control structure. The format of the illustration uses the notation of the Rumbaugh functional model (17:123-144).

This completes the description of the IVC design in particular and the overall *RDT* design in general. Chapter V discusses the implementation of these designs into the "C++" and "C" languages on Silicon Graphics workstations.

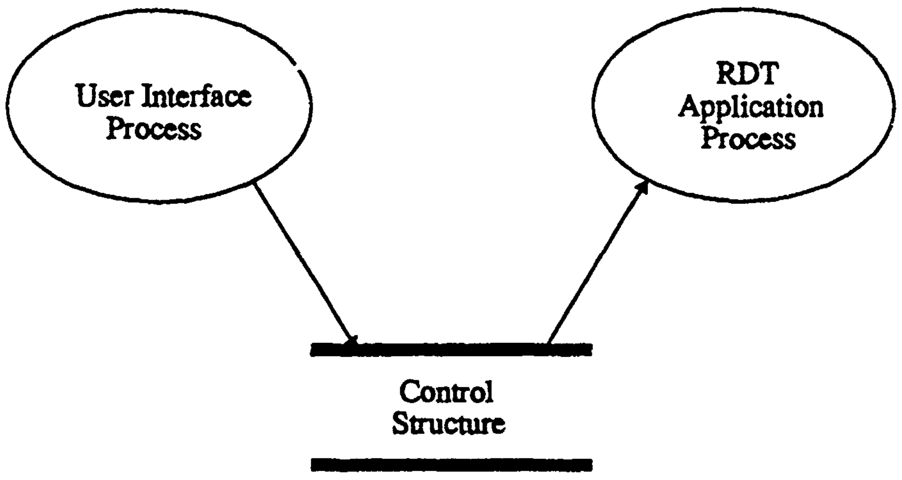

Figure 40. User Interface Integration

V. Implementation

Implementation is the translation of the design into a target language for a target machine. This chapter presents the details of the implementation process for each of the three major components defined earlier: CCC, DTC, and IVC. The organization of this chapter follows accordingly.

5.1 Computer Communications Component

The CCC is a combination of several serial devices and a single software program, called *Convert*, which perform a single high-level function–conversion and retransmission of RFMDS message blocks from the V.35 protocol into Ethernet. This section is divided into two subsections which describe the CCC's hardware and software elements.

5.1.1 Hardware. The final design of the CCC in section 3.1.2 identified the need for a "T" junction, a V.35 to RS-422 protocol converter, and a computer workstation. The actual hardware used to implement the "T" junction consists of two additional V.35 to RS-422 protocol converters.

Figure 41 portrays the full implementation of the CCC. Data arriving from the Pilkington fiber modem on the VAX computer side of the connection enters the first protocol converter where it is converted to RS-422. An A/B switch box is used to form the actual "T" and channels RS-422 data to the remaining two protocol converters. The converter, which is "in line" with the VAX link, restores the original V.35 signal for use by the VAX while the remaining converter is colocated with the computer workstation some distance away. RS-422 facilitates the data transmission over this moderate distance. This last converter reconstructs the V.35 signals from the RS-422 and makes them available to the computer workstation.

One of the early obstacles of the project was the lack of detailed information about the specific hardware protocols used by the RFMDS. The construction of an "active T" with three V.35 to RS-422 converters was a short term solution adapted to an an unknown environment. Future implementations may chose to limit the number of serial devices in the data pipeline by eliminating the third converter and performing the conversion in the

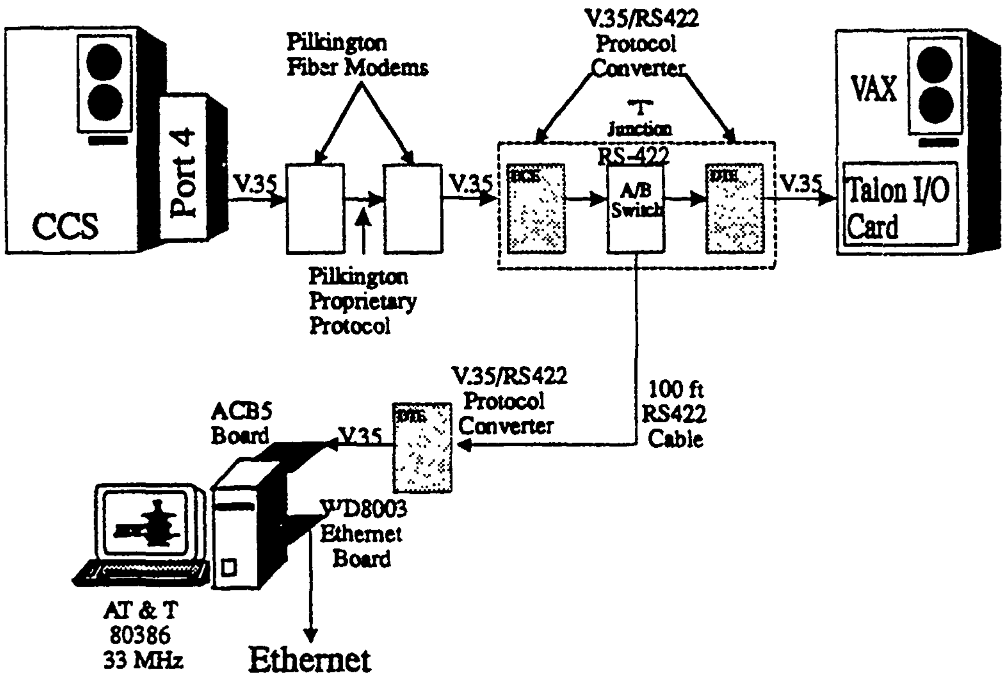

Figure 41. Hardware Implementation of the CCC

workstation from RS-422 to Ethernet as depicted in Figure 16. This modification reduces the risk of a device failure shutting down the entire pipeline by reducing the number of devices.

The selection of a computer workstation to convert V.35 to Ethernet was driven by the availability of suitable protocol converters and the desire to keep cost at a minimum. A communications board (ACB5) compatible with PC/XT/AT computers was located which could receive the V.35 data on one of two high-speed synchronous/asynchronous ports. This board is based upon the Intel 82530, AMD or Zilog 8530 serial communications controller chip. The specifications for the board limit support of the data rate to 1Megabit (Mb)/sec (8:4). This rate is insufficient for the 1.344Mb/sec data of the RFMDS; however, the Intel 82530 is rated at 1.5Mb/sec. To boost the data rate of the ACB5 board, the existing clock crystal was replaced with a 6MHz crystal. A 33MHz AT&T, 80386 compatible computer, available at AFIT , houses both the ACB5 board and a Western Digital Ethernet card. This combination of devices provides the following required capabilities for the CCC listed below.

1. Convert V.35 to Ethernet
2. Filter out unused data from the message blocks
3. Provide disk storage for mission data.

5.1.2 Software. The *Convert* program is the only software part of the CCC. *Convert*, written in the "C" language by Bruce Clay, a software engineer on staff at AFIT, incorporates an ACB5 device interface that was adapted from a low-level interface template provided with the ACB5 board.

Clarksen packet drivers from the public domain are used for the Ethernet network interface. A UDP/IP point-to-point protocol was chosen over TCP/IP because of uncertain timing requirements. It could not be readily determined whether the CCC workstation could use the TCP/IP protocol with its error checking and device acknowledgements and still maintain the high data rates of the RFMDS. (Data errors encountered during TCP/IP communications require that the data be retransmitted.) Numerous retransmissions could slow the CCC and cause incoming data from the RFMDS to be lost. Rather than design the CCC to guarantee that all packets arrive intact at the DTC, selection of the UDP/IP protocol optimized the transmision speed of the CCC.

The algorithm specified in section 3.1.3 provides the level of detail necessary to understand the implementation of *Convert*. The only detail of the implementation which needs to be emphasized is the difference between the message blocks sent from the RFMDS to the CCC and the individual RFMDS messages retransmitted by the CCC. A message blocks consists of one or more different, individual messages. Typically a message block contains the range time message first and then is followed by a maneuver data message, low activity data message, range status message and others as required or until the block is filled to capacity. The CCC parses the message block into the individual messages and sends each as a single logical Ethernet packet. Several modes are available as command line switches which allow data to be read from the V.35 port or from a file. An option is available to transmite data out the Ethernet port or save it to disk. See Appendix B for specific details about available command line options.

5.2 Data Translation Component

Unlike the previous component which consisted primarily of hardware devices, the DTC is principally software which relies on the hardware network interfaces to perform its task. The responsibility of the DTC is to accept individual RFMDS messages, translate the data from the RFMDS message formats into DIS PDUs, and transmit the PDUs onto a simulation network. The means whereby hardware and software were combined to complete these tasks are presented in the following subsections. The final subsection introduces the modificiations made to the Entity State and Detonation PDUs that are not in compliance with the existing DIS v2.0.3 draft standard.

5.2.1 Hardware. The target machine for the DTC was a Silicon Graphics workstation. The decision to use the SGI was based primarily upon the availability of SGI workstations in the AFIT graphics lab and familiarity with the SGI software suite and not on any hard requirement for a top-of-the-line graphics engine. Given the availability of a compatible Ethernet interface and a high- speed processor, any workstation could theoretically perform the translation task of the DTC. Section 6.1.4 discusses the performance issues relative to the use of single processor versus multi-processor machine for running the DTC software. Future hardware implementations of the DTC should examine the feasibility of running the CCC and DTC software on a single multi-processor workstation. This could further reduce the number of serial devices in the pipeline and decrease the risk of pipeline failure.

Additional hardware required to complete the DTC consists of a Delni fan-out box for Ethernet connections, two fiber-to-ether modems, and fiber optic/Ethernet cables. Figure 42 shows the addition of the DTC equipment to the CCC devices. The fiber-to-ether modems and fiber optic cable is used only to demonstrate the capability to transfer the data across high-speed communication lines, such as might be done on a commercial T-1 line used in wide area networks. Local use of the DTC only requires that Ethernet cables be connected from the Delni fan-out box to the IVC workstation.

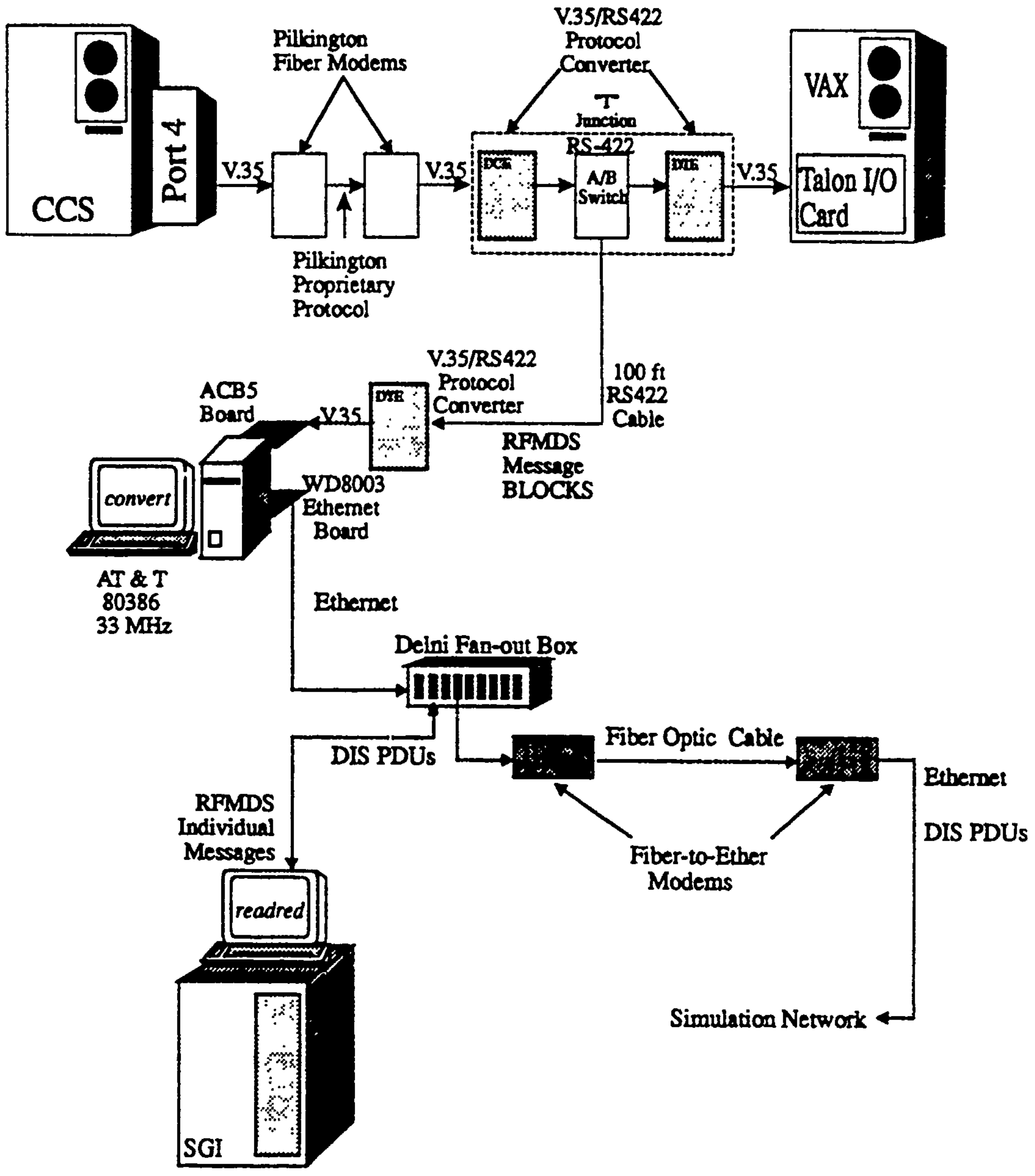

Figure 42. DTC Hardware Implementation

5.2.2 Software. The software for the DTC consists of two major components, a network daemon capable of broadcasting DIS PDUs and the *Readred* translation program. The daemon is the sole network interface for *Readred* and is described by Sheasby (18:47-57). (The daemon description (18) actually refers to an earlier version used with SIMNET. This daemon was modified to use the DIS protocol in subsequent versions and is currently undocumented. Some concepts of the SIMNET version still apply.) The algorithm design given in section 3.2.3.1 for the DTC provides sufficient detail to understand the structure of *Readred* with three notable exceptions–external control, internal data structures, and orientation determination for weapons.

5.2.2.1 External Control. *Readred* was originally written to allow a separate process to control the operation of the program through UNIX sockets. A socket is a special UNIX file type that allows client and server processes to pass data back and forth (24:28,261).

The motiviation for establishing an external interface to *Readred* was to allow a separate user interface to control *Readred's* operating modes (read from file, read from network), transmission speeds (start, stop, .5x, 1x, 2x, ...x times normal speed), and data filtering capabilities (translate info for all aircraft, or translate only a subset of aircraft.) The user-interface and *Readred* would together form a software element capable of replaying Red Flag missions from saved data files for aircrew analysis. Code exists within *Readred* to initialize the sockets and perform the communication, but this feature has not been fully implemented.

5.2.2.2 Internal Data Structures. *Readred's* internal data structures consist of fixed length buffers and arrays. These structures promote simplicity and speed in a stable RFMDS environment. Early software profile analysis of *Readred* revealed that more than 38% of the CPU cycles used during the translation of 1,000 blocks of data were spent performing floating point conversions between the RFMDS floating point format and the *IEEE* format. Further examination uncovered the fact that several of the floating point data fields from the RFMDS message buffers were accessed multiple times during the PDU generation process. This required expensive floating point conversions to be

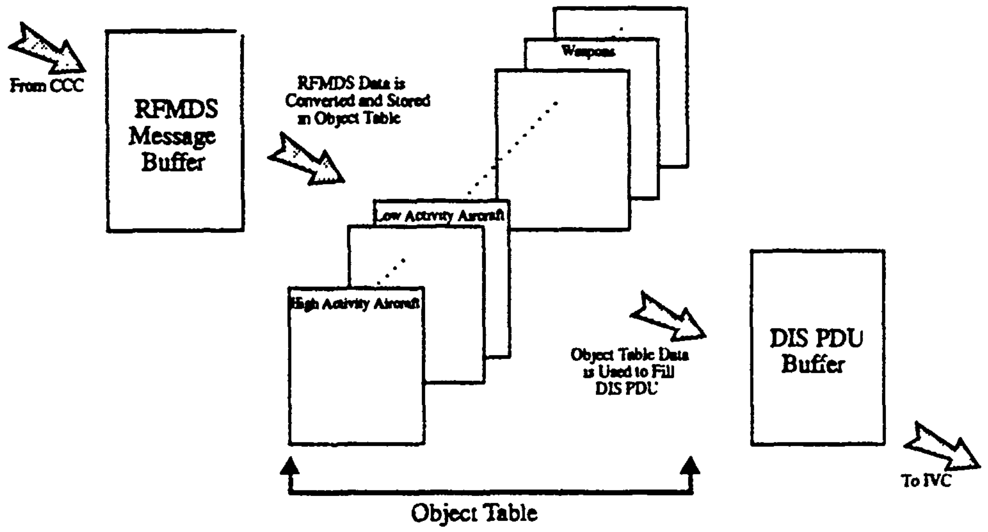

Figure 43. Readred Data Structures

performed several times for a single data item. The solution to the problem was to create an intermediate data structure, or table of objects, in which all of the floating point data could be converted once and then made available for other subroutines as necessary. This data structure was also used for storing identification information and previous position and velocity values for dead reckoning calculations.

Figure 43 illustrates the configuration of the arrays and buffers. Data received in the RFMDS message buffer is parsed and copied to the object table. The object table is a fixed length array with reserved elements. Elements 1 - 99 are held for high activity aircraft. (Growth potential is available since only 36 aircraft can be tracked as high activitiy aircraft at once.) Elements 100-199 are designated for low-activity aircraft. Weapons data is reserved for elements 200-249.

The RFMDS aircraft number is the key for storing aircraft data in the object table. For instance, data for high-activity aircraft number 36 is stored in object table element 36 and data for low-activity aircraft number 105 is stored in the object table at element 105. High-activity aircraft can only be differentiated from low-activity aircraft by their aircraft number, which equates to an RFMDS slot number. The logical player number, a unique number in the range of 1-255 that is assigned to each aircraft for an entire mission, does not distinguish between aircraft types. (Types must be discernable in order to correctly construct the DIS PDU.) All maneuver data messages use the aircraft number and not the logical player number as the key. In order to reduce constant mapping of the aircraft number to a logical player number and to keep a consistent interface for users familiar with the RFMDS numbering scheme, the aircraft number was retained as the key for referencing any aircraft. An additional detail that should be kept in mind when thinking about key values and the object table is the way in which RFMDS swaps high activity and low-activity participants. As a flight of high activity participants completes its mission and begins to depart the range, the slots used by an equal number of low and high-activity participants are swapped. This swapping permits all aircraft carrying AIS pods to potentially become high activity participants and have data recorded for their part of the mission. These 'slot swaps' are immediately preceeded by the transmission of new participant data messages. Consequently, the aircraft identities are updated in the object table prior to any reference to the new aircraft numbers and each participant's data is correctly stored.

Weapons are also considered to be objects and the their data is likewise stored in the object table. The key value is the weapon simulation slot number that ranges in value from 1 to 50. Using this number as the key value is a bit more complicated than with the aircraft numbers because RFMDS reuses weapon simulation slot numbers frequently. The RFMDS can only perform 50 simultaneous weapon simulations. Each weapon that is fired acquires the first avaialble RFMDS weapon slot. This slot number is unique among the 50 simulations but does *not* uniquely identify a single weapon over the entire length of the mission. For example the first weapon, an SA-2 missile, launched at an aircraft would occupy slot 1 in the RFMDS weapons data message. A weapon launched while the SA-2 is still active, would occupy slot 2. After the SA-2 detonates, slot 1 is made available and the next weapon to be launched would immediately reuse slot 1. The key used for weapons is 200 + the slot number. In order to uniquely indentify a weapon for the entire exercise a master weapon identification number is assigned by *Readred* to each new weapon upon receipt of a fire signal. This master weapon ID number is stored in the object table and used as the weapon's entity number in the DIS PDU. Master weapon numbers are in the range of 256 to 400. These beginning and ending values are tied to the player numbers in

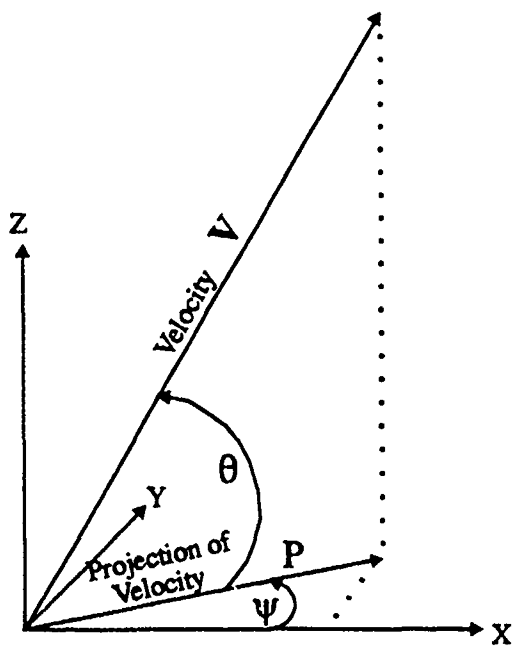

Figure 44. Heading and Pitch Determinations from Velocity

the IVC. Weapon numbers are reused if 400 is exceeded. A large range in the weapon ID number makes it extremely unlikely that two weapons with the same weapon ID number will be active at the same time.

5.2.2.3 Orientation Determination for Weapons. RFMDS weapon simulation data contains only the location of the weapon and is void of any orientation information. It is possible to compute a weapon's heading and pitch from two sequential position updates. The components of the velocity vector V are determined by:

$$V_x = \frac{(x_1 - x_0)}{(t_1 - t_0)} \tag{4}$$

$$V_y = \frac{(y_1 - y_0)}{(t_1 - t_0)} \tag{5}$$

$$V_z = \frac{(z_1 - z_0)}{(t_1 - t_0)} \tag{6}$$

Figure 44 shows an arbitrary velocity vector in a right handed coordinate system. The projection of V onto the x, y plane forms the projection vector **P**. The angle ψ between the x axis and $\mathbf{P}$ corresponds to the rotation of $\mathbf{P}$ around the z axis beginning from the x-axis.

$$\psi = \arctan\left(\frac{V_y}{V_x}\right) \tag{7}$$

The angle ψ is in the range of $\pm 180°$ and must now be adjusted to represent the weapon heading in the range of $0 - 360°$. Headings correspond to clockwise rotations about the z axis beginning from the y axis. A common correction factor can be applied for all quadrants except Quadrant *II*, which requires a unique correction.

$$\begin{array}{rc} 90 - \psi & \textit{Quadrants I, III, IV} \\ 450 - \psi & \textit{Quadrant II} \end{array}$$

The angle θ formed between the projection vector $\mathbf{P}$ and the velocity vector $\mathbf{V}$ corresponds to a rotation about the x axis and represents a weapon's pitch in the range of $\pm 90°$. No corrections are needed and θ can be used directly.

$$\theta = \arctan\left(\frac{\|\mathbf{V}\|}{\|\mathbf{P}\|}\right) \tag{8}$$

5.2.3 Entity State and Detonation PDU Modifications. Subsection 3.2.3.3 identifies a number of fields in the Entity State and Detonation PDUs that are defined as padding or unused fields in the v2.0.3 draft standard but which are used by *RDT*. The need for a Zulu time stamp has already been addressed; however, three other issues remain: replay speed, capabilities, and the reason behind a hit or miss for a weapon simulation.

5.2.3.1 Dead Reckoning Record Modification. A companion issue to the need for a Zulu time stamp is the requirement for a replay capability using the DIS PDUs. Replay of an exercise is not addressed in the DIS standard but is the key activity upon which all mission debriefs are based. Immediately upon return from a mission, flights review their individual aircraft video tapes and extract key events. Each of the mission forces (Blue Air, Red Air, and Blue Air-to Ground) then gathers and conducts a mission review using the RFMDS to put the pieces of the puzzle together into an accurate representation of the unique mission activities. Aircrews use RFMDS variable replay speeds to minimize the mission review time and to slow down complex engagements so that a complete picture of the fight is formed. A mass debrief with all participants is the last item on the agenda. The RFMDS is used once more to replay the mission. Variable replay speeds are used to skip the non-essential mission elements identified in previous reviews and provide 'stop action' dissection of the major battles. The RFMDS and its displays are the equivalent of the television network's instant replays.

An additional 4-byte floating-point data element was placed into the unused portion of the 120-bit dead reckoning parameter fields of the Entity State PDU at byte offset 4. This element, the replay speed, is an indicator to other applications that the current PDU is being broadcast in a replay mode at the speed indicated. Dead reckoning procedures at the receiving end of the DTC have to modify their algorithms to account for the replay speed by multiplying their time increments by the replay speed in the PDU. This is necessary because the velocities in the PDUs remain fixed. *Readred* sets the replay speed equal to 1.0 for live operations or normal replay, but is also capable of using a simulation-speed parameter in its time synchronization loop. Applying dead reckoning in a replay mode within *Readred* is one of many areas for further research and testing.

5.2.3.2 Capabilities Record Modification. Weapon and range status information is encoded within the RFMDS maneuver data message in the weapons, status and itrace fields. Boolean weapon status bits in the maneuver data message indicate the following (3:A-71):

1. Radar lockon
2. IR missile lockon
3. IR missile seeker uncaged
4. Aircraft dead

In a traditional DIS exercise, a detonation PDU is broadcast by the weapon entity's simulation application. It is the responsibility of the target entity's simulation application

Bit 7 MSB	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 LSB
Filter Predict	Filter Unreliable	Laser On	IR Uncage	Aircraft Dead	No Slot	IR Lockon	Radar Lockon

Table 5. Weapon and Status Byte

to determine the extent of the damage caused by the detonation and report any damage in the appearance field of the Entity State PDU. The roles of both applications in the *RDT* environment are satisfied by the RFMDS. The aircraft dead bit is used to signal a successful weapon engagement and is therefore placed into the Entity State PDU so that the IVC can render an appropriate image. The radar and IR bits are passed to the IVC but are not currently used to modify the rendering of aircraft images.

Filters used to smooth raw data from the AIS pods and also detect anomalies in TIS tracking report the quality of the telemetry information in two ways. First, a set of status bits is used to indicate the operating mode of the filters (normal or prediction) and the reliability of the filtered data. Second, itrace status codes show conditions of filter internal operation.

These codes indicate a range of operations including: tracking aided by radar altimeter, downlink data consistency failure, uplink data failure, aircraft off range, and many others (3:A-73). Any abnormal aircraft movements depicted by the IVC can be cross-checked against the integrity of the data being transmitted for the aircraft. This is a valuable feature often used to answer questions about apparent misorientation. Table 5 shows how the weapons and status bits are composited into a single status byte.

While the force ID field of the Entity State PDU contains an 8-bit enumerated value that can be used to associate a unique color for each of the defined forces, the DIS v2.0.3 draft standard does not define a field which identifies the role of an entity, such as defensive counter-air, interdiction, air refueling, stand-off jamming, etc. Aircraft role information is encoded in the RFMDS displays with unique colors. In this manner, aircraft performing the same role can be readily identified because they appear with the same color. The following roles are encoded into the *RDT* capabilities record:

1. AAR - Air to Air Refueling
2. ABCCC - Airborne Communication Command and Control
3. ADF - Air Defense
4. AWAC - Airborne Warning and Control
5. BAI - Battlefield Air Interdiction
6. CAP - Close Air Patrol
7. CAS - Close Air Support
8. CIJ - Close In Jamming
9. CMJ - Communications Jamming
10. DCS - Defensive Counter Air
11. ECM - Electronic Counter Measures
12. HELO - Helicoptor Operations
13. INTR - Interdiction
14. NUC - Nuclear
15. OCA - Offensive Counter Air
16. REC - Reconnasance
17. FAM - Familiarization
18. SAR - Search and Rescue
19. SEAD - Suppression of Enemy Air Defenses
20. SOJ - Standoff Jamming

Table 6 depicts the content and format of the 4-byte *RDT* capabilities record that replaces the DIS 32-bit boolean capabilites record.

Byte 3 MSB	Byte 2	Byte 1	Byte 0 LSB
Itrace	Weapon/ Status Byte	Mission Role	Unused

Table 6. RDT Capabilities Record (4 Bytes)

5.2.3.3 Detonation PDU Modification. The Detonation PDU defined in the draft standard uses an 8-bit enumated type to describe the results of a detonation but fails to provide a field that can be used to identify the reason a particular weapon engagement failed. While this data may not be available in the DIS world, it is transmited in the weapons data messages of the RFMDS simulations. To aid the user in shot analysis, the miss reason is passed on to the IVC in the detonation PDU. The 16-bit padding field following the number of articulated parameters at the end of the PDU is filled with the miss reason.

5.3 Interface and Visualization Component

Now that the hardware configuration has been fully described up to the point of connecting the IVC machine to the network, it is time to illustrate this connection in terms of the overall implementation and also show how data flows through the different software components. Subsection 5.3.1 contains the two figures that tie all of the components together into a complete system. In order to provide needed background information for the class method discussions in subsection 5.3.3, additional information is presented in subsection 5.3.2 which highlights *Performer's* multi-processor and shared memory requirements. Subsection 5.3.4 addresses specific problems and solutions used within the IVC to provide more realistic images. The final subsection illustrates the results of the design and implementation efforts with a set of photos from a typical *RDT* session.

5.3.1 RDT Hardware and Software Overview. Figure 45 illustrates the complete *RDT* system. A comparison with Figure 42 from section 5.2.1 reveals that a single SGI workstation is added as the platform for the IVC software. The IVC platform requires a top-of-the-line graphics engine in order to produce images representing the dynamic aircraft activities of a Red Flag mission at a rate that provides smooth aircraft movements. (The desired update rate was 15 frames per second for Z-buffered, flat-shaded polygons.)

Figure 46 provides a similar overview of the *RDT* but from a software/data-flow viewpoint. From an abstract level of observation, the data flow can be viewed as a pipeline. The live RFMDS data flows into the pipeline at the top of the diagram and flows out the bottom of the pipe into the IVC component for image generation. Removing the layer of abstraction uncovers the interplay between data and the software components. Live RFMDS data is received by the *Convert* program at the top of the figure. The data is converted to Ethernet packets and transmitted to the *Readred* program for translation to DIS. *Readred* uses the AFIT DIS daemon to broadcast the DIS PDUs onto a distributed simulation network. RDT_{dis}, the IVC component, uses the AFIT DIS daemon to monitor the network and retrieve DIS PDUs for processing by the Object manager. The Object Manager supplies player position updates and weapon events to RDT_{dis} from the object manager.

The flow of data through the pipeline is somewhat different if a mission is to be replayed from a data storage device, such as a tape or disk. A magnetic tape containing the RFMDS message blocks captured by the VAX computer can be converted to a disk file using resources at AFIT. (A single 47 minute Red Flag mission uses approximately 180 Megabytes of storage.) The raw data file can then be used directly by the *Convert* program.

Readred on the other hand expects individual RFMDS messages and cannot directly process the raw data file. During live broadcast, the *Convert* program extracts only the individual RFMDS messages from the blocks and discards the unused data. This same preprocessing step must be done with the raw data file before *Readred* can interpret the data correctly. There are two ways to produce this preprocessed file:

1. Use *Convert* to store the extracted messages to disk

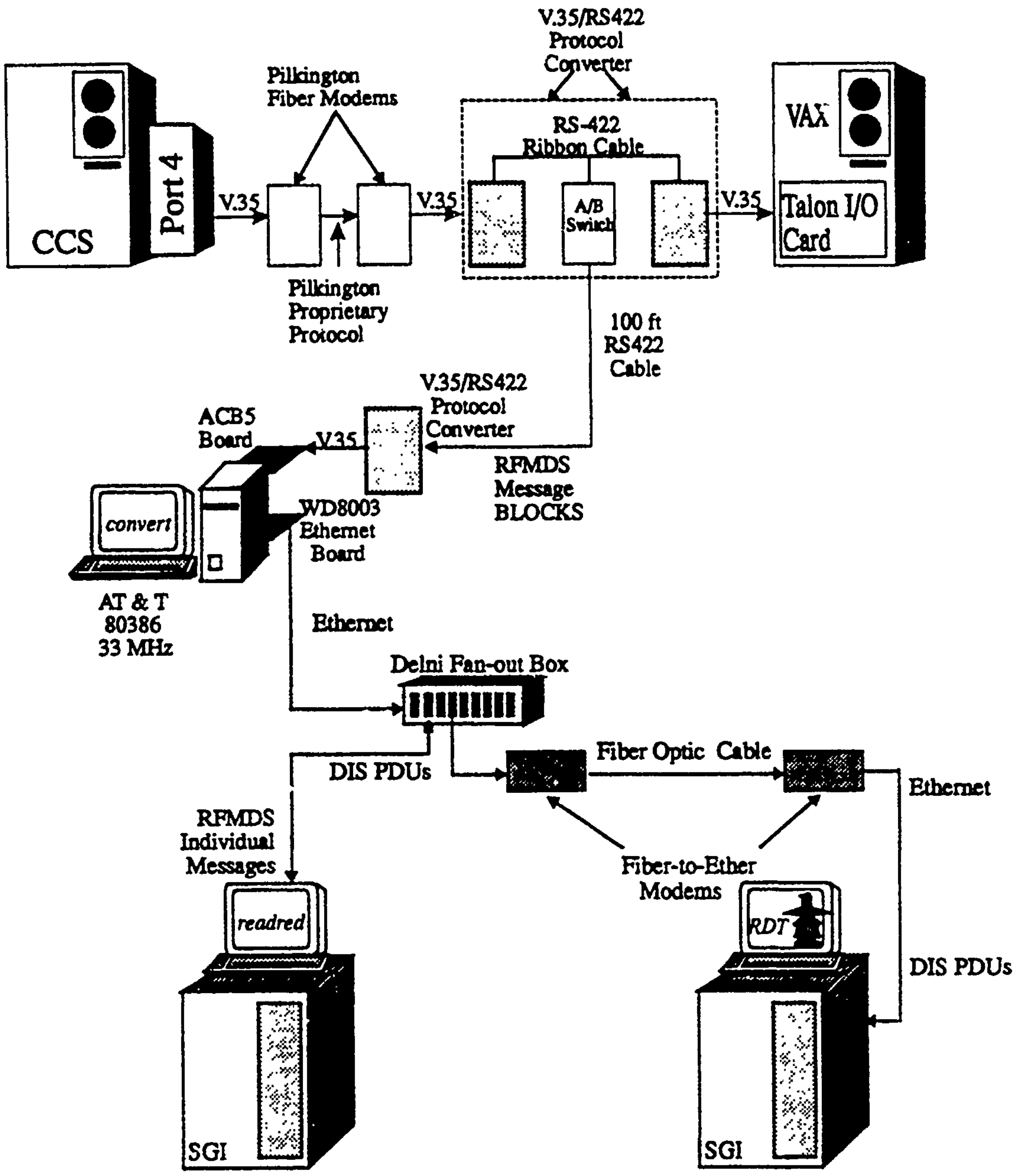

Figure 45. Complete RDT Hardware Implementation

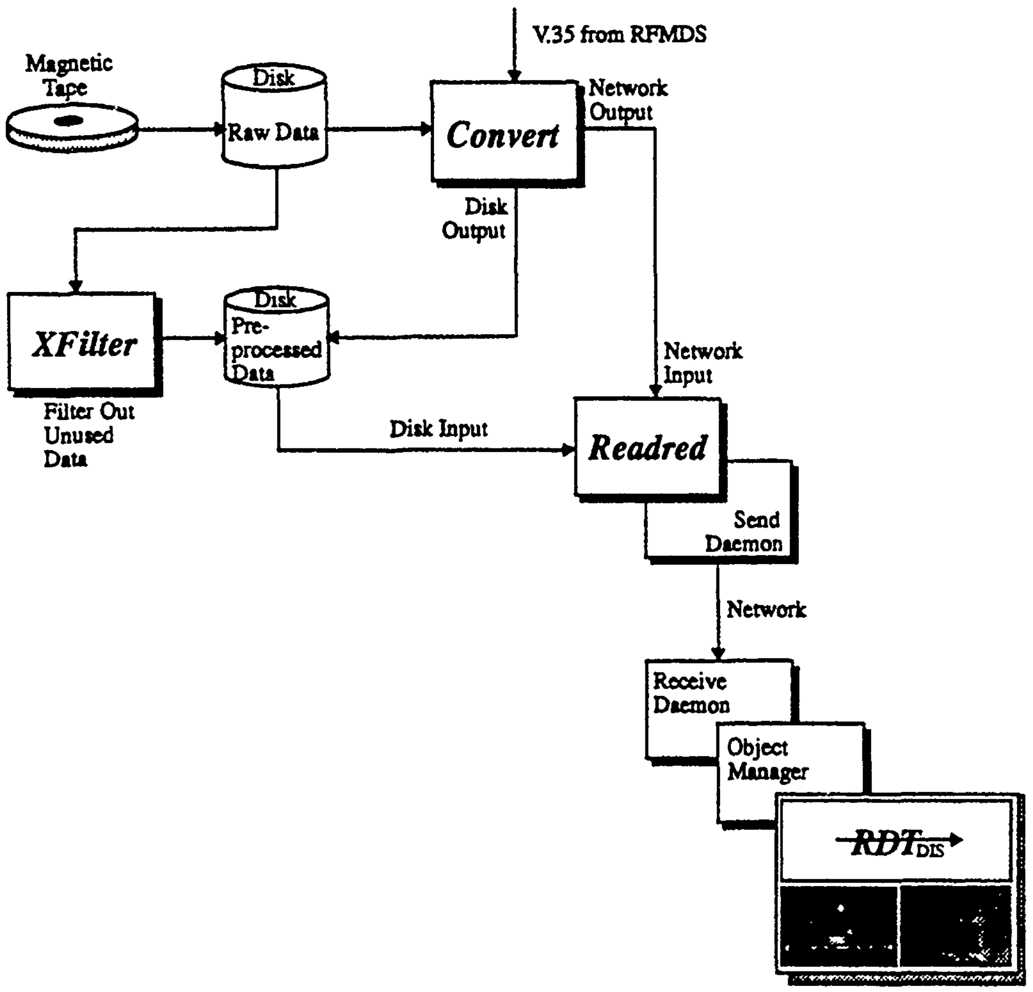

Figure 46. RDT Software Components and the Data Pipeline

2. Use the *Xfilter* program to process a raw data file and produce the required filtered message file. (See Appendix D for details concerning the operation of the *Xfilter* program.)

Either method creates a data file suitable for use by *Readred.* The data-flow is identical once the *Readred* program begins translation.

5.3.2 Performer Shared Memory and Multi-Processing. Visual simulations which use the SGI *Performer* software have a "built-in" capability to access multiple processors, if they are available on the machine, because of the way in which *Performer* partitions the functional stages of its graphics pipeline. The three stages of the graphics pipeline are the APP, or application stage, the CULL and the DRAW stages. In the APP stage, the simulation application interacts with other networked stations and propagates the positions and orientations of its models through the scene. The CULL stage traverses the visual database of models and determines which polygons are visible in the current view volume. Selection of appropriate models for level-of-detail switching also occurs in the CULL stage. The DRAW stage issues the graphics library commands to the geometry pipeline in order to produce the image on the display (11:7-13). The work performed in each of these stages can be distributed between up to three processors. Using three processors, each allocated to one of the functional stages of the graphics pipeline, provides the maximum throughput for rendering model geometries.

When *Performer* operates in the multi-processor mode, each pipeline stage is allotted a full frame period to perform its work, and it takes two additional frame periods for a model movement computed in the APP stage to be reflected in the final image produced by the DRAW stage. This condition is generally not noticeable unless the application performs its own draw functions in the APP stage onto geometry drawn by *Performer* in the DRAW stage. Rapid position changes of the models can highlight this frame latency and cause unwanted distractions. For example, assume that the application desires to draw a tail number on an aircraft model in the APP stage. As long as the model position does not change rapidly from frame to frame, the tail number will appear at the desired location on the model being drawn in the DRAW stage. However, if the position changes rapidly from one frame to the next, the tail number may be drawn well ahead of the aircraft. By keeping a history of the location data for three consecutive frames and using the values that the DRAW stage will use, the latency problem can be overcome. This technique has not been incorporated into the current version of the IVC.

Process "callbacks" allow applications to insert their own custom culling and draw functions into the rendering pipeline. RDT_{dis} uses the post-draw "callbacks" to draw the aircraft numbers, flight path trails, and the cockpit HUD into the scene after *Performer* has drawn the scene geometry.

Data needed in multiple stages of the pipeline must reside in shared memory in order to be visible to the other processes. For example, an aircraft's position that is updated in the APP stage running on one processor is also needed in the DRAW stage on another processor so that flight path trails can be drawn. Forms "callbacks" from the user-interface are made during the DRAW stage. Information which these "callbacks" process to control the views of the IVC is stored in shared memory so that the APP stage of the application can have access to the data and modify the view parameters accordingly. Shared memory and process visibility to common data are crucial concepts in the multi-processor RDT_{dis} environment.

5.3.3 Class Method Descriptions. The following subsections detail RDT_{dis} specifics by describing the data structures and methods invoked by the RDT_{dis} Net Manager, Event Manager, Player and User-Interface classes. Other classes derived from the *ObjectSim* framework or newly created for RDT_{dis} have straight forward methods and interfaces and do not require additional explanations beyond those given in sections 4.2.1 and 4.2.2. (In the following descriptions the term "entity" is synonomous with a DIS entity and "player" is the term used for an entity in the RDT_{dis} environment.)

5.3.3.1 Net Manager. The Net Manager class is the primary interface between RDT_{dis}, the Object Manager, and the simulation network. The Object Manager is a separate process which operates concurrently with RDT_{dis}. The **init** method is used to spawn the Object Manager process. Once created, the Object Manager continuously reads PDUs from the network, archives entity data, and dead reckons entity positions.

```
while (TRUE)
{
  update entity information from the network;
  dead reckon each active entity;
}
```

During the Object Manager's update procedure the daemons are polled to determine if new PDUs have arrived. If new PDUs are available, the data is incorporated into the Object Manager's class structures and arrays. The dead reckoning procedure determines the time differential between the current time and the previous dead reckoning call and applies the dead reckoning algorithms to each of the active entity's coordinates. Orientation dead reckoning is not implemented.

The Net Manager interrupts this continuous update/dead reckon cycle during each rendering frame with the **update** method. The **update** method receives **from** the Object Manager the number of active players and their identities and passes to the Object Manager the location of the RDT_{dis} player array. The Object Manager places the new PDU information for each entity directly into its corresponding player.

A formal protocol between the Object Manager and Net Manager defines how entities become active players in RDT_{dis}. The Object Manager changes a player's state from "deactive" to "inserted" the first time that an Entity State PDU is received for that player. The Net Manager changes the player state from "inserted" to "active" and inserts a new model of the required type into the scene at the new player's coordinates. Thereafter, each update from the Object Manager changes the player's coordinates as a result of either dead reckoning or a PDU update.

Player coordinates, orientation and velocities received from the Object Manager conform to the DIS v2.0.3 standard. The RDT_{dis} software, much like the RFMDS software, was written using a simple flat-earth or tangent-plane world coordinate reference model. A set of utility programs written by Erichsen (4) changes the DIS round-earth coordinates, orientation, and velocities into the flat-earth frame of reference used by RDT_{dis} and all of the other *ObjectSim* applications developed at AFIT during 1993. It is the interaction of the Net Manager and Object Manager that propagates players through the RDT_{dis} scene.

Active players are removed from the scene by a similar formal protocol. Once the Object Manager determines that an entity is no longer active, the state of the corresponding player is changed from "active" to "deleted." The Net Manager then removes the corresponding model from the scene and changes the player's state from "deleted" to "existed". The "existed" state is used to differentiate a player that has never been active from one that has been active and since been deleted.

5.3.3.2 Event Manager. The Event Manager has an interface with the Object Manager similar to that of the Net Manager. The Event Manager passes to the Object Manager the location of its event list and receives **from** the Object Manager the number of new events that the Object Manager read from the network and placed into the Event Manager's event list.

Each event is placed into a circular queue. Events can later be retrieved in reverse chronological order with Event Queue methods. DIS round-earth coordinates and velocities are also converted to the flat-earth reference frame with the same set of utilities used by the Net Manager.

Along with each fire and detonation event, the bearing, range, delta altitude and closing velocity between the launcher/weapon and the target are computed and stored. The following equations define how this data is computed.

The range between the launcher/weapon located at (x, y, z) and the target located at (x_t, y_t, z_t) is:

$$Range = \sqrt{(x_t - x)^2 + (y_t - y)^2 + (z_t - z)^2} \quad (9)$$

The difference between the launcher/weapon altitudes and the target altitudes is simply the difference in their z coordinates.

$$Delta\ Altitude = z_t - z \quad (10)$$

The closing velocity between the launcher/weapon at the moment a weapon is fired or detonated is computed in a three-step process. First, the line-of-sight vector, **LOS**, between the two players is calculated and then normalized, **nLOS**.

$$\begin{aligned}
LOS_x &= x_t - x && (11)\\
LOS_y &= y_t - y && (12)\\
LOS_z &= z_t - z && (13)\\
nLOS_x &= \frac{LOS_x}{\|\mathbf{LOS}\|} && (14)\\
nLOS_y &= \frac{LOS_y}{\|\mathbf{LOS}\|} && (15)\\
nLOS_z &= \frac{LOS_z}{\|\mathbf{LOS}\|} && (16)
\end{aligned}$$

Next, the velocity vector difference, $\triangle$ **V**, between the launcher/weapon velocity, **V**, and target velocity, $\mathbf{V}^t$, is computed.

$$\begin{aligned}
\triangle V_x &= V_x^t - V_x && (17)\\
\triangle V_y &= V_y^t - V_y && (18)\\
\triangle V_z &= V_z^t - V_z && (19)
\end{aligned}$$

Finally, the closure velocity, V_c, is the projection of $\triangle$ **V** upon **nLOS** and is in the direction of **LOS**.

$$\begin{aligned}
V_c &= \mathbf{nLOS} \cdot \triangle\mathbf{V} && (20)\\
&= \|\mathbf{nLOS}\| \; \|\triangle\mathbf{V}\| \; \cos\theta\\
&= \|\triangle\mathbf{V}\| \; \cos\theta
\end{aligned}$$

The bearing from the launcher/weapon to the target is computed identically to the procedure outlined in section 5.2.2.3.

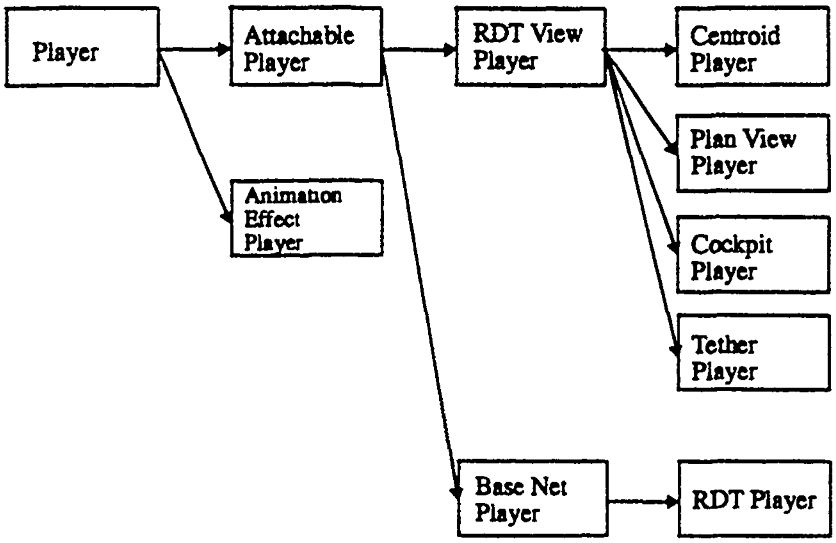

Figure 47. Player Class with Derivations

In the event that Fire PDUs are transmitted but not received, weapon entities become *RDT*$_{dis}$ players upon receipt of their Entity State PDUs and are propagated along their flight path by the Net Manager. If no Detonation PDU is received, the weapon player will continue to be dead reckoned until the Object Manager determines that it is no longer active and deletes the weapon.

5.3.3.3 Player. *RDT*$_{dis}$ players are derived from the *ObjectSim* player class through multiple layers of inheritance. Two classes are defined–the RDT Player and the RDT View Player. Figure 47 shows the derivations for each of these player classes from the base class, Player. Both classes are Attachable Players but are distinguished from each other by their data structures and methods.

RDT Player Class. The RDT Player is void of any methods except those which manage the flight path history information in the player's circular flight path queue. The RDT player is basically a repository of state information that is used by the *ObjectSim* Renderer and the User Interface displays. The RDT Player attributes are:

- RFMDS Time Stamp
- Force ID
- Call Sign
- Abbreviated Call Sign

- Aircraft Type
- Velocity Vector
- Angle of Attack
- Angle of Side Slip
- Rate of Climb
- Crab Angle
- Mach Number
- Indicated Airspeed
- True Airspeed
- G Force
- Mission Role (Color)
- Override Color
- Radar Lock Flag
- IR Missile Lock Flag
- IR Missile Uncaged Flag
- Laser On Flag
- Number of Radar Missiles Fired
- Number of Successful Radar Shots
- Number of IR Missiles Fired
- Number of Successful IR Missile Shots
- Number of Gun Shots
- Number of Successful Gun Shots
- Number of Times Killed
- Flight Path History Circular Queue
- Pointer to the 3D Model
- RFMDS Data Filter Unreliable Flag
- RFMDS Data Filter Predict Mode Flag

Attributes that are derived from the base Player class and Attachable Player class are not shown. One of these attributes, however, is a model pointer. Two instances of the same model geometry are used for each RDT Player. Both models are inserted into and removed from the scene at the same time by the Net Manager, but only one model is visible in a view at a time. View masks are associated with each model so that one model is visible only in the Plan View and the other model is visible only in the 3D View. By creating

Element Number	Category
0	Bullseye Player
1 - 255	Red Flag Players
256 - 400	Red Flag Weapons
401 - 500	Network Entities

Table 7. RDT Player Array Indices

two separate models, each can be scaled independently. This allows the model rendered in the Plan View to be one size while the model drawn in the 3D View is another size. Model scaling is performed by the RDT View Players so that the player's orientation and location are always visible in the Plan and Centroid Views. This artificiality is necessary to build a view of the overall range airspace and participants which enhances situational awareness.

A static array of RDT Players is created during initialization. The first element of the array, Player 0, is used for the Bullseye player. Elements 1 - 255 are reserved for Red Flag players and elements 256 - 400 are reserved for Red Flag munition entities. Elements 401 and beyond are reserved for network players from sources other than Red Flag. Table 7 summarizes these array index assignments. This first implementation is not optimized for dynamic scaling of the simulation such as drastically increasing the number of entities in the simulation; however, it does provide a simple and efficient means of randomly accessing any player received from the network.

RDT View Player Class. Four RDT View Players are defined in RDT_{dis}: Plan, Centroid, Cockpit, and Tether. Each RDT View Player encapsulates similar methods, propagate and draw, which are differentiated by the way a view player's location is determined or the type of additional information drawn into the scene.

Plan View Player. The Plan View Player is propagated through the RDT_{dis} scene by changing the x, y, and z coordinates with the control panel interface. The Plan View player is oriented such that the view is always in the direction of the negative z axis, or looking down. RDT Player models are scaled so that they are always visible.

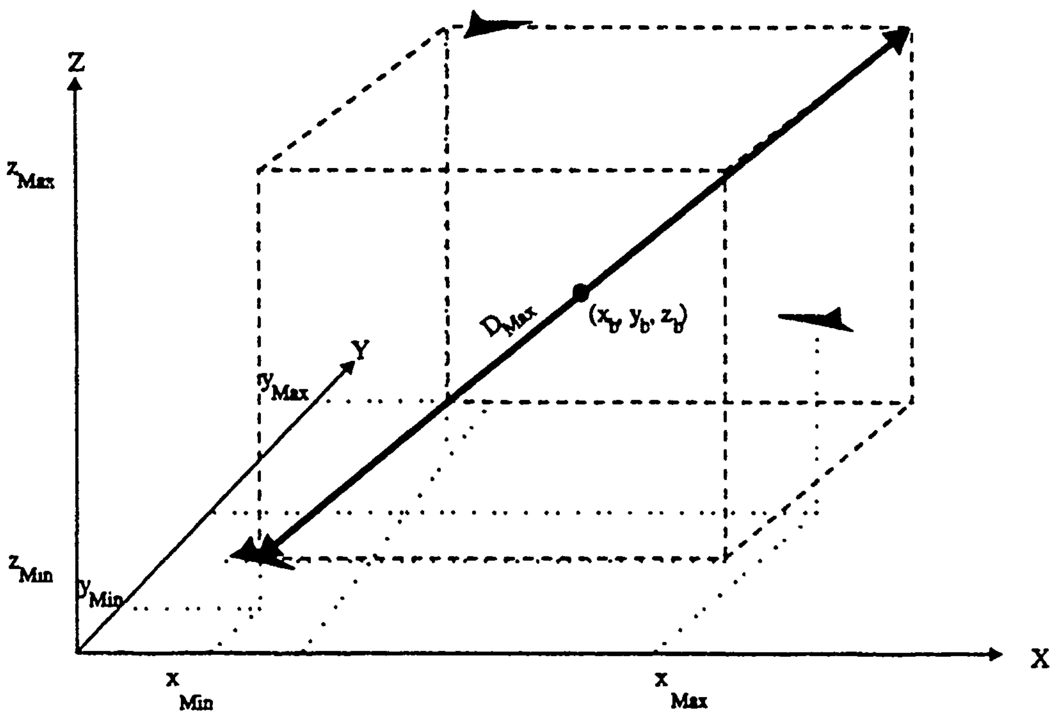

Figure 48. Bounding Box Determination for Centroid View

Cockpit and Tether View Players. Both the Cockpit and Tether View Players are indirectly propagated by the Net Manager because these players are attached to one of the active RDT Players. The Tether View player is positioned a fixed distance behind the RDT Player on an extension of the velocity vector. The Cockpit Player is positioned a fixed distance forward of the RDT Player's location. Model scaling is not performed so that the view from the cockpit or tethered trail position reflects reality as much as possible.

Centroid View Player. The Centroid View Player's position is dependent upon the number of selected RDT Players, their locations, the view elevation, and view heading. The default Centroid View Player position for a single RDT Player is a fixed distance above the player looking in the direction of the negative z-axis. A multi-step procedure is used to determine the Centroid View Player' position and view orientation when more than one RDT Player has been selected.

First, RDT_{dis} determines a default overhead view position and direction by calculating a bounding box which encompasses all of the selected RDT Players (also referred to as centroid players), and the Centroid View Player's final position (x_c, y_c, z_c). Figure 48 illustrates a bounding box containing three centroid players. The coordinates (x_b, y_b, z_b), the center of the bounding box, are defined as:

$$x_b = (\frac{x_{Max} - x_{Min}}{2}) + x_{Min} \tag{21}$$
$$y_b = (\frac{y_{Max} - y_{Min}}{2}) + y_{Min} \tag{22}$$
$$z_b = (\frac{z_{Max} - z_{Min}}{2}) + z_{Min} \tag{23}$$

The maximum separation that can exist between any of the centroid players is less than or equal to the distance between opposite corners of the bounding box. This maximum separation distance, D_{Max}, can be used to determine a position above (x_b, y_b, z_b) so that the viewing volume contains all of the centroid players. The distance from z_{Max} to (x_c, y_c, z_c) is defined as D_R.

$$D_R = Ratio_{FOV} * AspectScale * \left(\frac{D_{Max}}{2}\right) \tag{24}$$
$$Ratio_{FOV} = \frac{\cos(FOV)}{\sin(FOV)} \tag{25}$$

$Ratio_{FOV}$ is a scale factor which is used increase/decrease D_R depending upon the field of view. If a large field of view is used, say 120°, D_R can be decreased so that the view position is closer to the centroid players than if a smaller field of view, 90°, is used.

AspectScale is an additional multiplier which not only increases D_R so that the centroid players do not appear on the very edge of the viewing volume, but also adjusts for the rectangular aspect ratios of most monitors. *AspectScale* was experimentally derived and varies dependent upon the largest of two ranges–x_{Min} to x_{Max} or y_{Min} to y_{Max}. *AspectScale* is largest when the y_{Min} to y_{Max} range is the greatest.

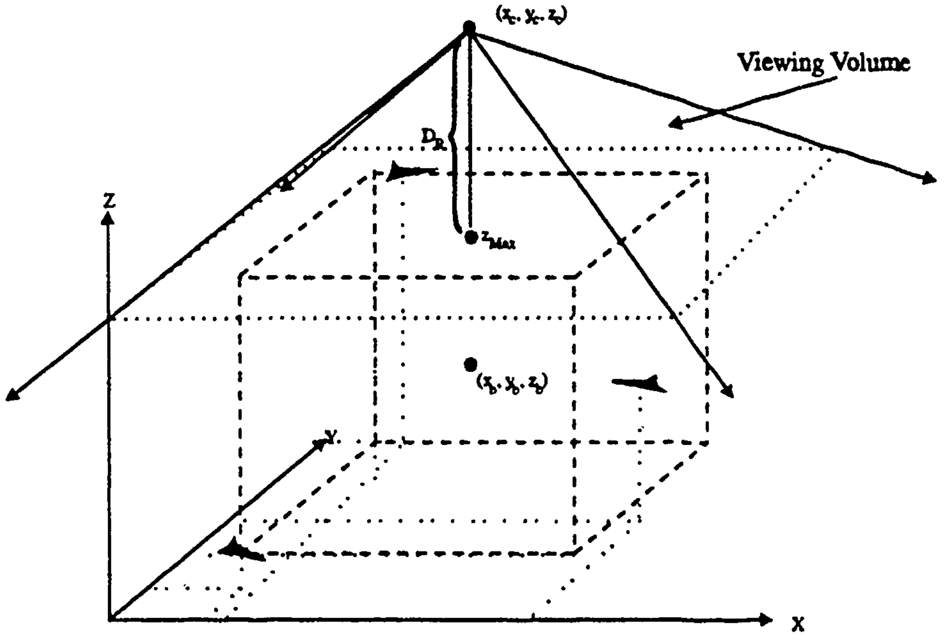

Figure 49. Default Centroid View Player Position with Multiple RDT Players

The default Centroid View Player's view orientation is in the direction of the negative z-axis and the position, shown in Figure 49, is defined as:

$$(x_c, y_c, z_c) = (x_b, y_b, z_{Max} + D_R) \tag{26}$$

Once the default position is found, the position and view direction can be modified to conform to the heading and elevation inputs from the user-interface control panel. A horizontal slider on the control panel is used to select the view heading. If the slider value is 090 then the view direction should be oriented so that the Centroid View Player's heading is 090°, or looking East. Similarly, a vertical slider on the control panel is used to select the view elevation. The elevation may range from 90°, overhead view, to 0°, side view, as depicted in Figure 50.

By controlling the Centroid View Player's elevation and heading with the control panel inputs, (x_c, y_c, z_c) is moved over the surface of a hemisphere located at (x_b, y_b, z_b)

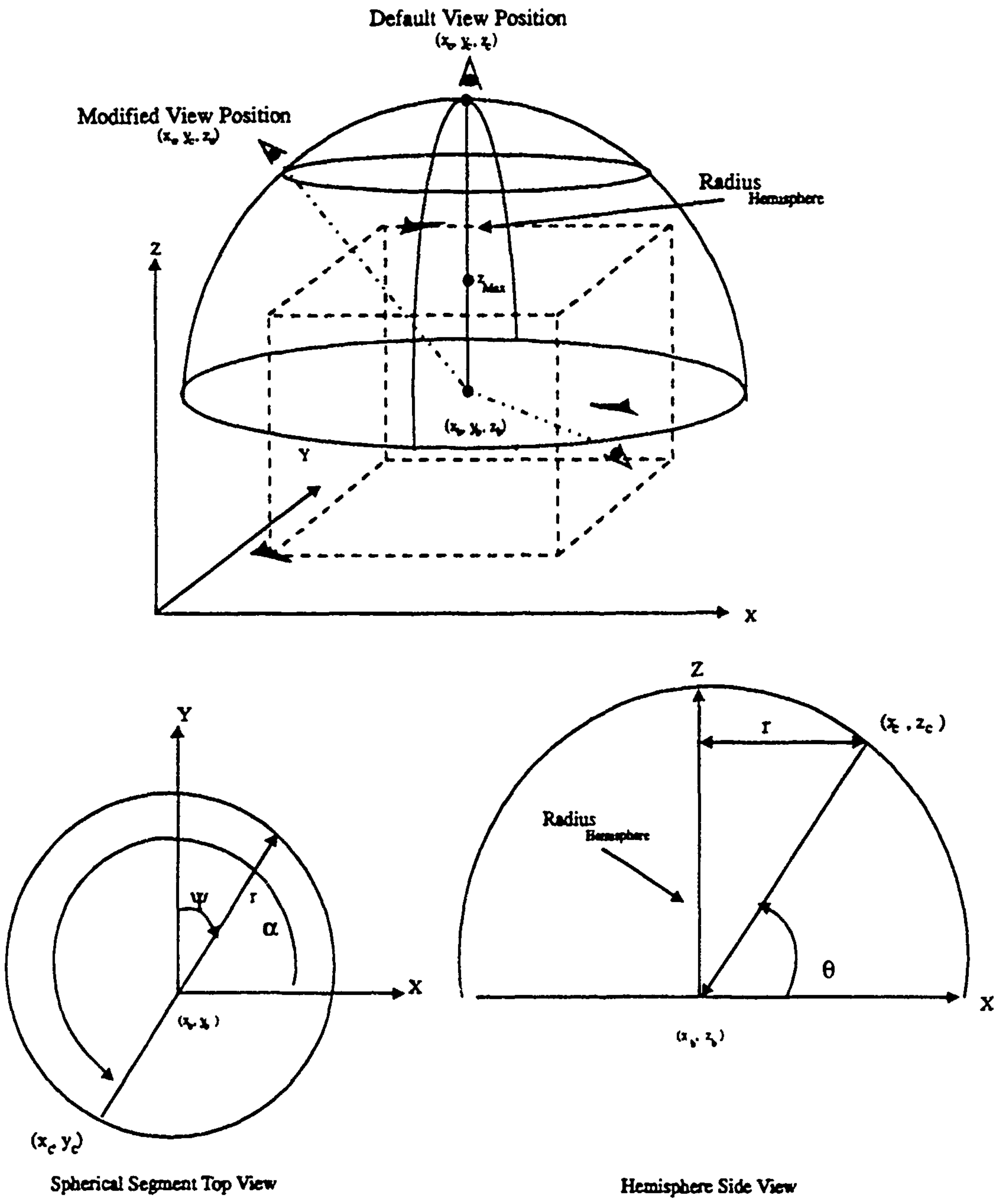

Figure 50. Centroid View Player Elevation and Direction

with radius, $Radius_{Hemisphere}$. The view direction is from (x_c, y_c, z_c) towards (x_b, y_b, z_b).

$$Radius_{Hemisphere} = D_R + \left(\frac{z_{Max} - z_{Min}}{2}\right) \tag{27}$$

To determine x_c, y_c, and z_c, reference the top and side views of Figure 50. The heading input from the control panel, ψ, must first be converted from a negative rotation beginning at the positive y-axis about the z-axis to a positive rotation beginning from the positive x-axis about the z-axis. In addition, (x_c, y_c) needs to be positioned such that the final view direction corresponds to the control panel input. Let α be the desired angle.

$$\alpha = 270 - \psi \tag{28}$$

The radius, r, of the spherical segment at height z_c of the modified view position is defined as:

$$r = Radius_{Hemisphere} * \cos(\theta) \tag{29}$$

where θ is the elevation angle from the control panel input. (x_c, y_c, z_c) can now be calculated.

$$x_c = x_b + r\cos\alpha \tag{30}$$

$$y_c = y_b + r\sin\alpha \tag{31}$$

$$z_c = z_b + Radius_{Hemisphere}\sin\theta \tag{32}$$

5.3.3.4 User Interface. Although the user interface class in the object diagram of Chapter IV indicates a single class, the RDT_{dis} user interface is a conglomeration of "C" functions managed by the "C++" User Interface class. All user interface functions required within the RDT_{dis} application reference this class and its methods to perform their required tasks. Figure 51 illustrates the numerous source code files which comprise the user interface.

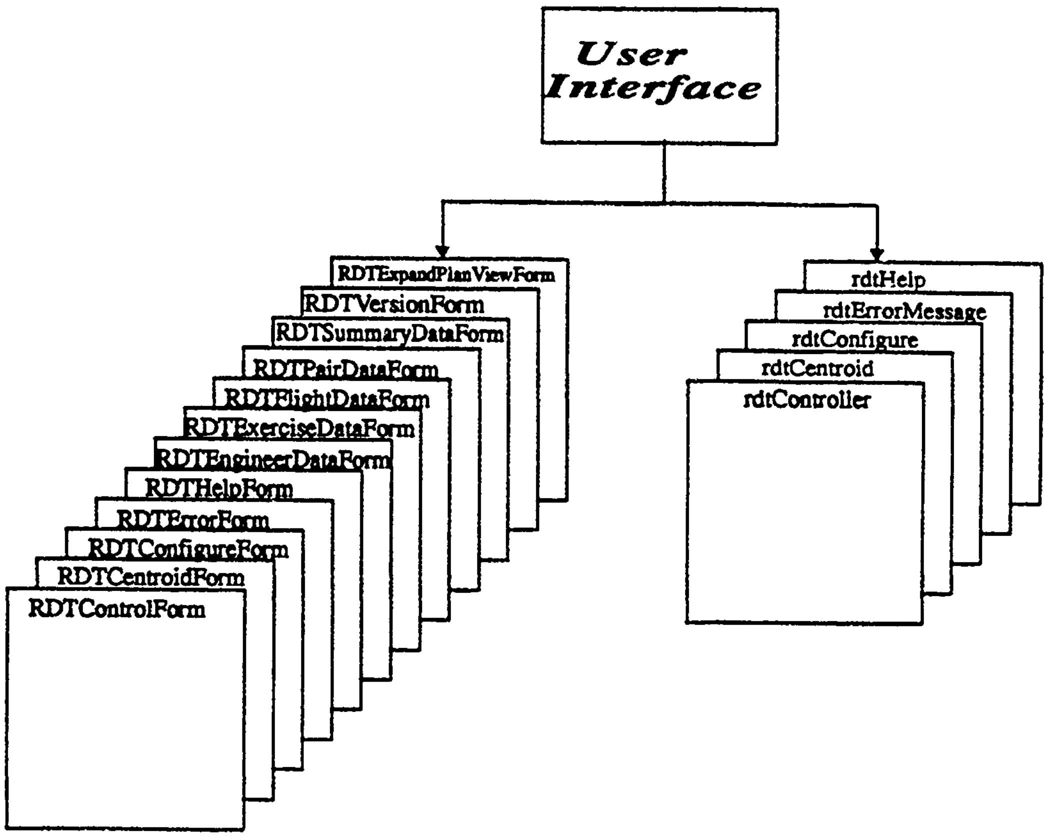

Figure 51. User Interface Software Modules

Item	LOC	Percent
Forms	7,979	29.8
Call backs	4,460	16.8
Configuration Files	1,300	4.8
Visualization	12,298	48.6
Total	26,737	100

Table 8. RDT Souce Lines-of-Code Breakdown

One of the features of the RDT_{dis} application and its interface that distinguishes it from other DIS stealth viewers is its ability to present low level data that can be used for detailed mission analysis. Stealth viewers allow a user to move through a simulation environment without interacting with any of the simulation players. Each of the graphical views available in RDT_{dis} could easily be found in one form or another in a stealth viewer application; however, the large volume of data passed from RFMDS to RDT_{dis} provides a level of detail unavailable in the majority of the DIS stealth viewer applications. It is precisely this characteristic which makes RDT_{dis} a viable tool for mission monitoring and debriefing.

The user interface comprises 46.6% of the more than 26,000 lines of code in the RDT_{dis} software. Table 8 illustrates the breakdown of the RDT_{dis} software in terms of lines-of-code (LOC). Considering only the user interface software, 64% was automatically generated by the *Forms Designer*. Only the semantic action procedures or "callbacks" needed to be coded "by hand." The high percentage of code that was machine generated speaks highly of the sophistication and flexibility of the *Forms Designer* software and its suitability for interface design and implementation.

5.3.4 Image Problem Resolution. Implementation of the IVC design on the Silicon Graphics machines was not a problem-free task. Early attempts to use elevation shading with the digital terrain data resulted in a terrain image that conveyed less information than the original RFMDS line drawings. This subsection details how a suitable texture map was created to alleviate this problem and also identifies one of the z-buffer vi-

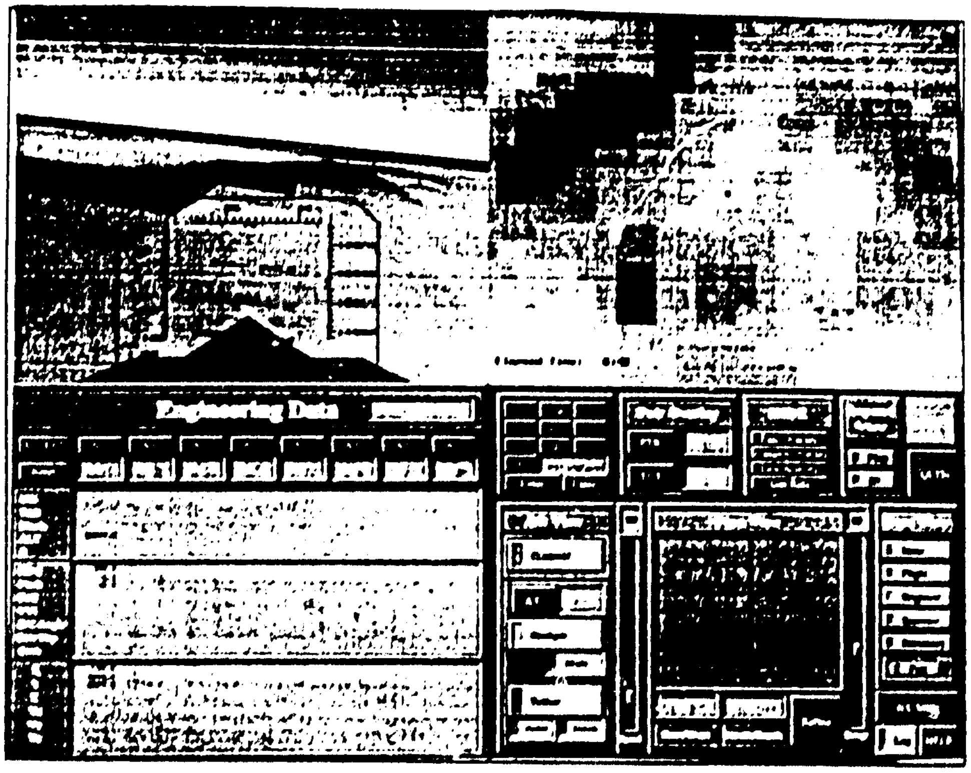

Figure 52. Early Elevation Shaded Red Flag Terrain

sualization artifacts that significantly detracted from the overall presentation and describes how this problem was resolved.

5.3.4.1 Terrain Texture Mapping. The photo in Figure 52 shows an early implementation of the digitized Red Flag terrain. Each square is composed of two triangles of slightly varying shades that are colored according the the elevation of the terrain represented by the triangles. Darker colors correspond to higher terrain. It is apparent from the cockpit view in the figure that the aircraft is approaching higher terrain, but the exact boundary between mountain and valley is unclear and does not reflect reality. The Plan View in the figure also shows the major features of the terrain, mountains and lowlands, but is void of any tangible details that truly coincide with the real terrain. Both views are unsuitable for use in a mission monitoring/debriefing tool where pilots navigate using terrain features and fly low to the ground. It would be difficult to credibly warn a pilot of an approaching hill or obstruction using the images depicted in the figure.

One of the driving forces in computer graphics image generators is the number of polygons in the scene which must be rendered. Clearly, a large number of polygons requires more time to render than a few simple polygons. The Red Flag terrain used in RDT_{dis} covers an area approximately 143 nautical miles wide by 120 nautical miles high. The number of polygons which would be needed to accurately represent this terrain would severly limit the number of images which could be rendered in a given amount of time. Slow frame rates translate to jerky, unresponsive simulations which only increase user dissatisfaction.

Texture mapping techniques exist which "paste" two dimensional texture patterns onto the polygons used in the scene. This provides a way to place a significant amount of additional information into the scene with fewer polygons. For example, a single polygon with a texture map taken from a detailed aeronautical chart could easily meet all of the needs of the Plan View in RDT_{dis}. The processing cost to render the single, texture-mapped polygon would be minor compared to a terrain data base composed of a million polygons, as long as the hardware was optimized for texture mapping. Silicion Graphics advertizes the new Reality Engine^2 Onyx computers with the capability to draw 320 million textured, anti-aliased pixels per second. These graphics super-computers, available at AFIT, provide the needed image generation speed to use texture mapping techniques to reduce polygonal complexity in the terrain data base and increase the amount of information available to a viewer.

Compare the photo of Figure 52 with any of the photos in section 5.3.5. Clearly visible is the marked difference between the early elevation shaded polygons and the texture mapped polygons in the final implementation. The 12 texture mapped polygons used in the lowest level of detail in the Plan View clearly shows sharply defined terrain features and offers the minimum polygonal scene complexity. These improvements yield the fastest image generation rate.

In order to create the texture maps used in RDT_{dis}, a high fidelty geometric model of each of the six 1° latitude by 1° longitude cells of the Nellis ranges was created. A desert color scheme for the model were chosen to correspond to ranges in elevation much the same way that the early terrain was shaded. *MultiGen* software from Software Systems, Inc. was used to render the elevation shaded geometry. A screen capture utility was used to generate an image of the high fidelty terrain. Evident in the image were the fine shaded triangles of the geometric model. To remove this artificiality, the snapshot image was blurred. The blurred image was then mapped to the polygons of the terrain to create a view into the simulation which accurately reflects the features found on the Nellis ranges.

5.3.4.2 Image Flicker. A hardware visible-surface determination technique called z-buffering is used in the Silicon Graphics machines to render realistic 3D images. This techique provides a simple and efficient method for determining which polygons, or portions thereof, are visible in a scene. A z-buffer with the same number of entries as the frame buffer is used to store the current depth, or z coordinate value, of each pixel. As new polygons are scan converted into the frame buffer, the depth value of each pixel is compared to the current depth value in the z-buffer. If the value is greater than the z-buffer value, the pixel is ignored. This situation occurs when the pixel in the current scan-converted polygon is farther from the view point than a previous pixel. If the depth value of the new pixel is less than the z-buffer depth value, then the new pixel must be in front of previous pixels and the depth and color information of the new pixel replaces the frame and z-buffer contents (5:668).

As the distance from the the viewpoint to overlapping polygons increases, the relative distance between the overlapping polygons grows smaller and smaller. Eventually, depth comparisons between the pixels of the overlapping polygons may be difficult, if not impossible, to distinguish because of the fixed numerical precision of the computer hardware/software. This difficulty manifests itself in the computer image as flickering. Pixels from first one of the overlapping polygons, and then others, randomly occupy the pixel locations in the frame buffer. In RDT_{dis}, aircraft which were flying near the surface of the earth often had random parts of their geometry replaced by terrain shaded pixels making it difficult to determine an aircraft's orientation.

In order to resolve this problem, the z-buffering techniques were disabled during the rendering of the Plan and Centroid Views. Although this may produce an image where

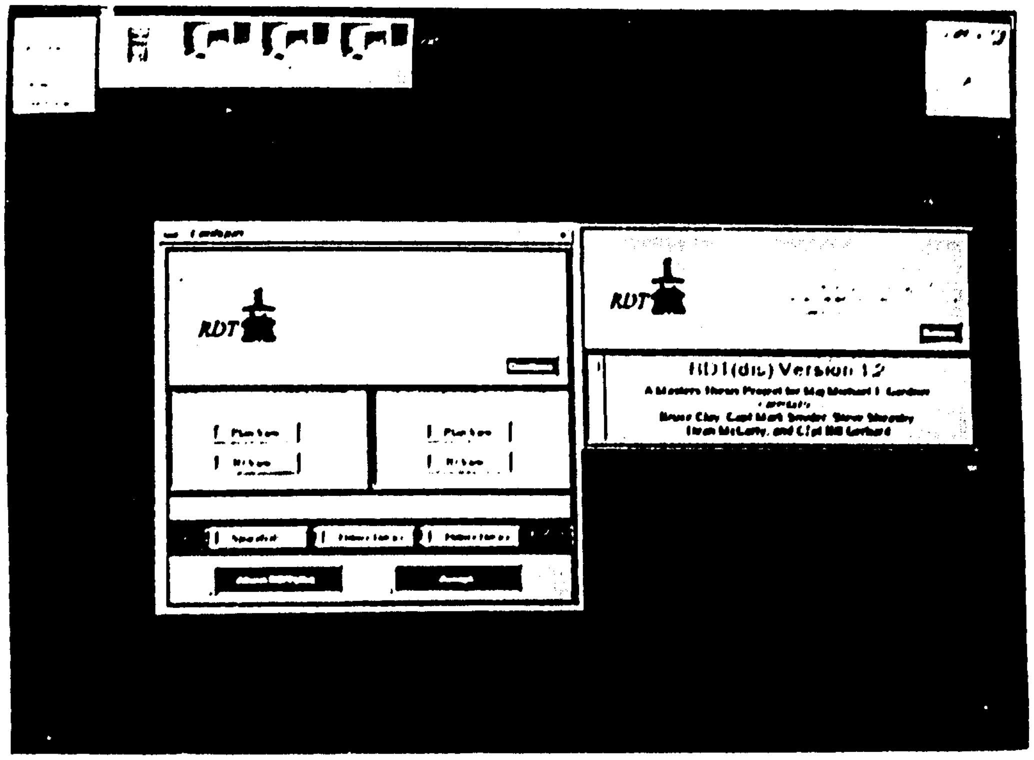

Figure 53. Initial Configuration Options

one aircraft incorrectly obscures another in the scene, the benefits derived from disabling z-buffering far outweigh a periodic error in the image.

The Cockpit and Tether views keep the z-buffering enabled so that they always depict an acccurate scene. In these two views, the flickering problem is less pronounced because the distance from the view point to any of the players is relatively small compared to the extreme distances seen in the Plan View.

5.3.5 IVC Photos. The following pages contain photos from a session with RDT_{dis}. The photo sequence illustrates the major capabilities of RDT_{dis} and the final form of the user interface. The next chapter presents issues related to *RDT* performance and compares the capabilities of RDT_{dis} with those of RFMDS.

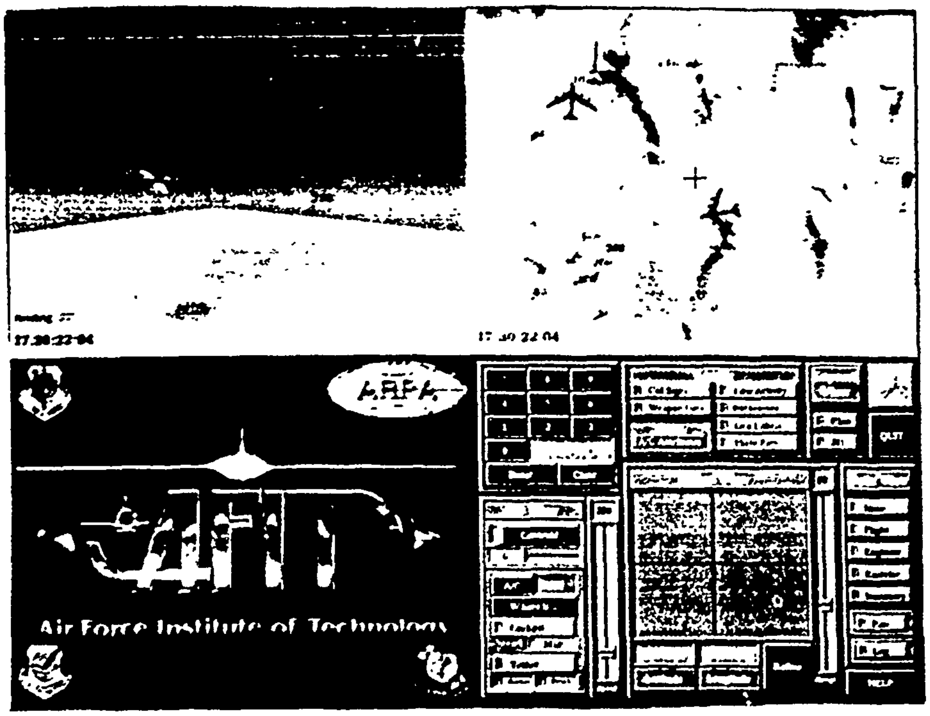

Figure 54. Simultaneous 3D and Plan Views

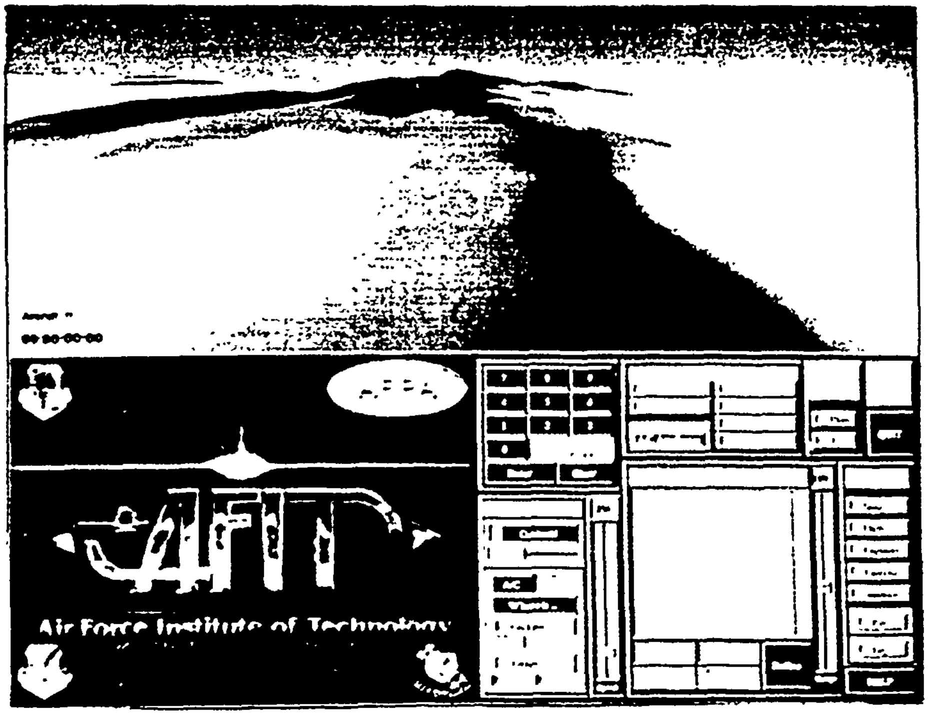

Figure 55. RDT Terrain with Texture Maps

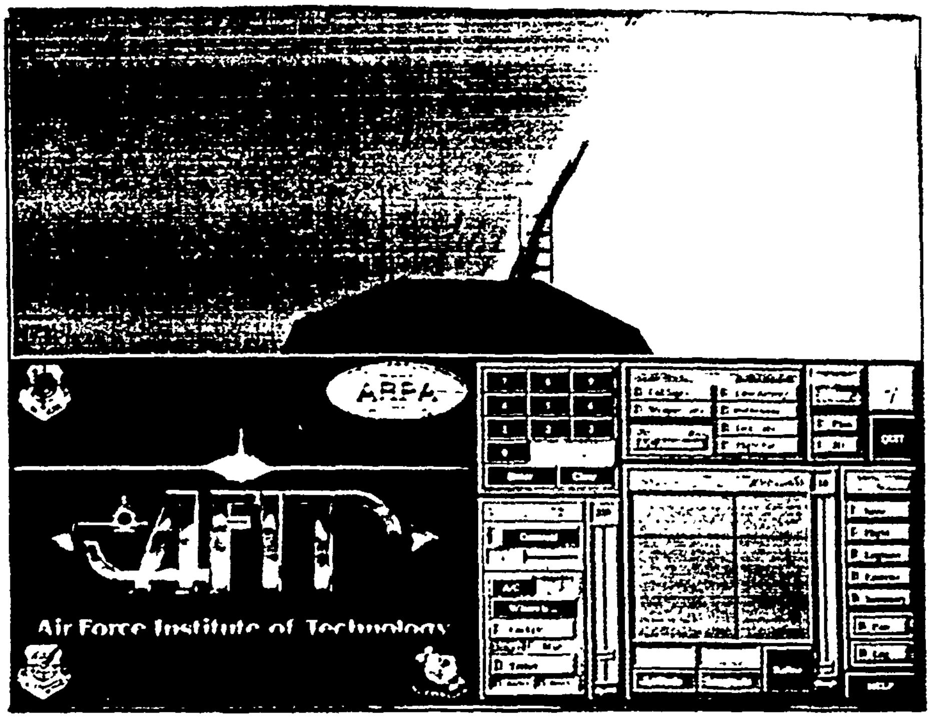

Figure 56. Expanded Cockpit View

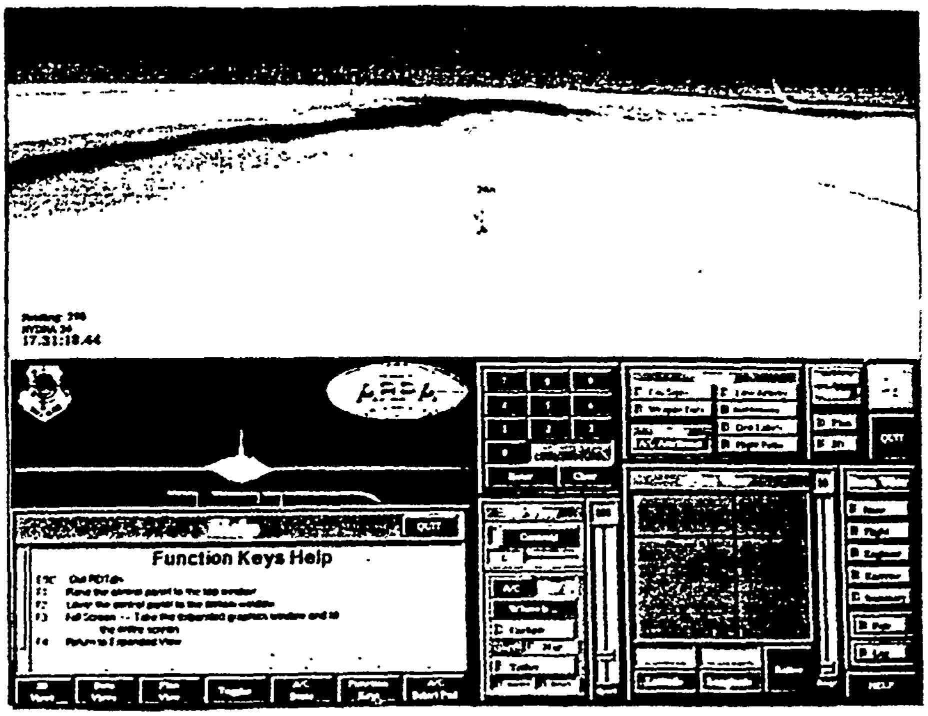

Figure 57. Expanded Tether View with Help Window

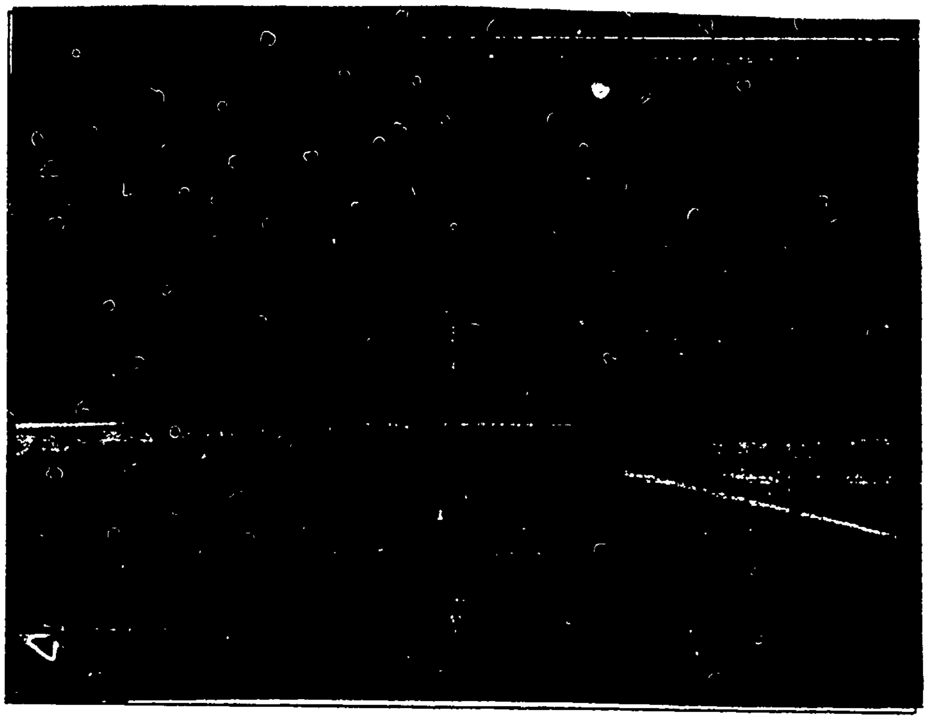

Figure 58. Full Screen Tether View

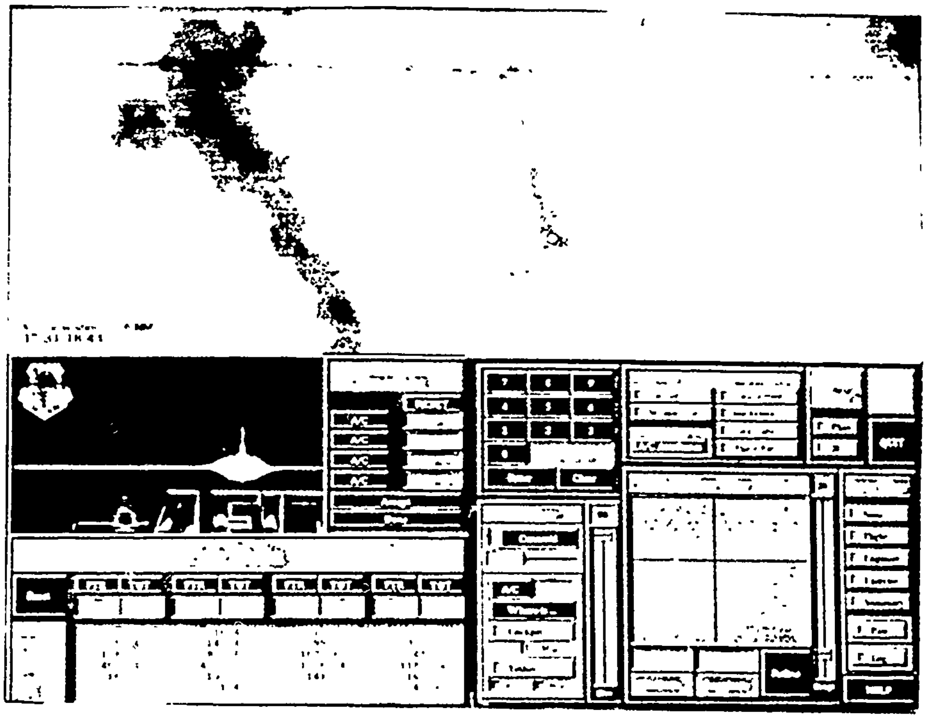

Figure 59. Expanded Centroid View with Bullseye and Pair Data

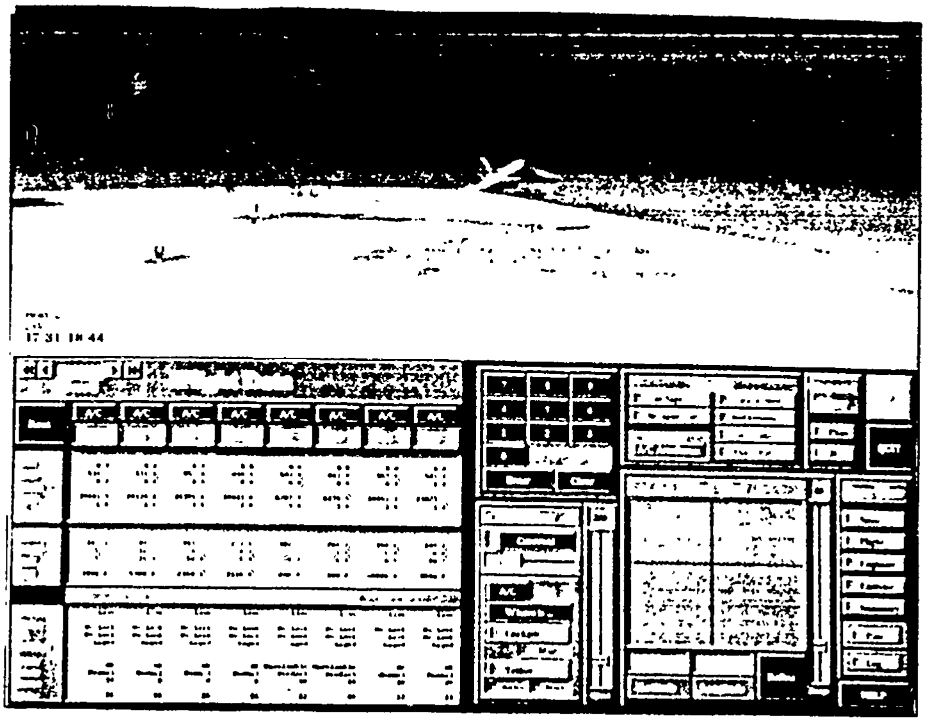
Figure 60. Flight Data

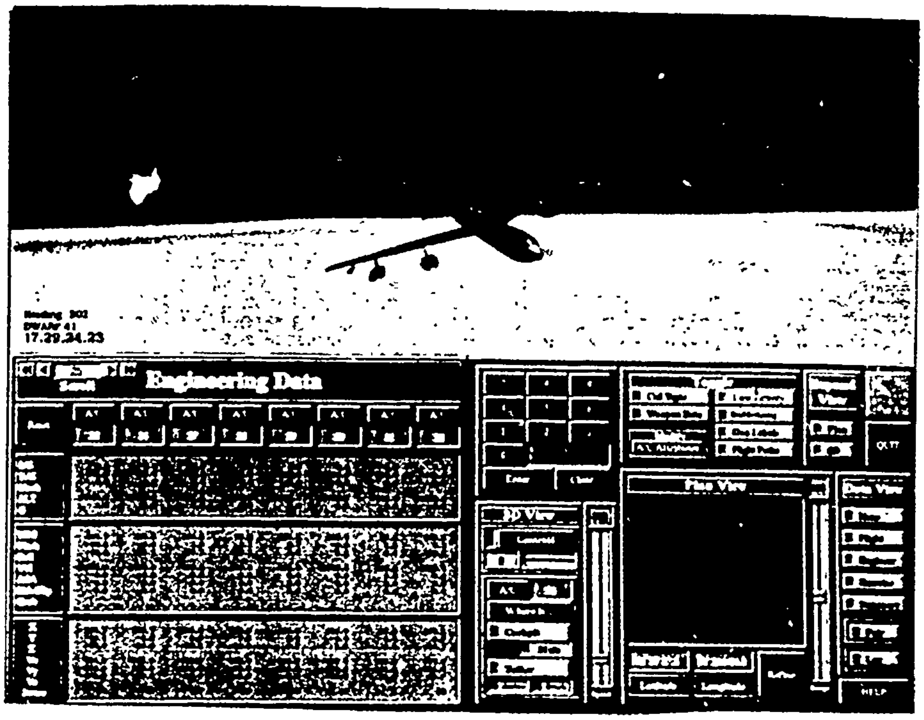

Figure 61. Engineering Data

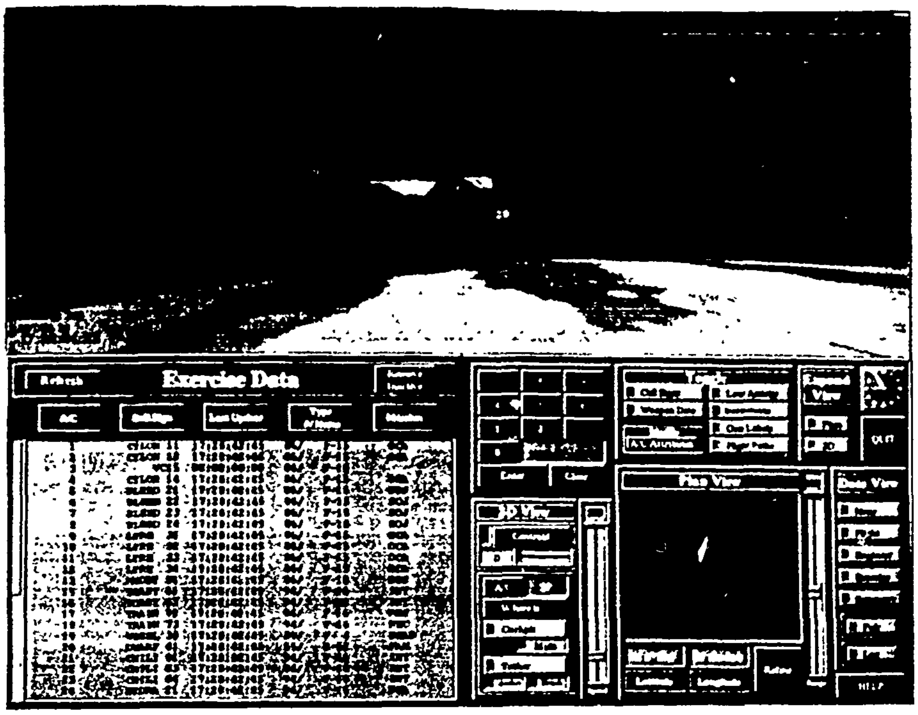

Figure 62. Exercise Data

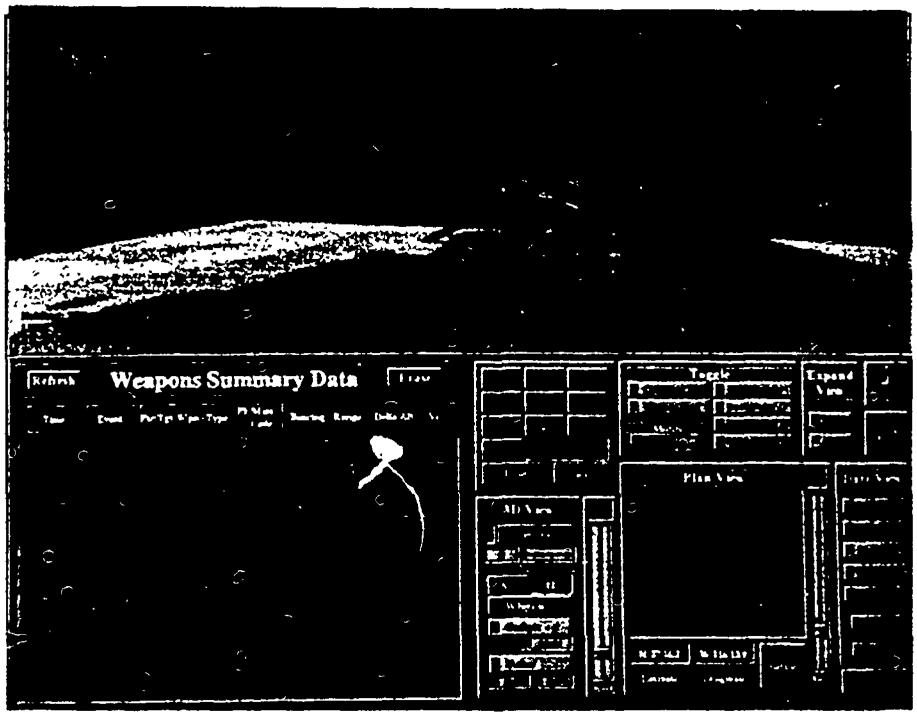

Figure 63. Summary Data

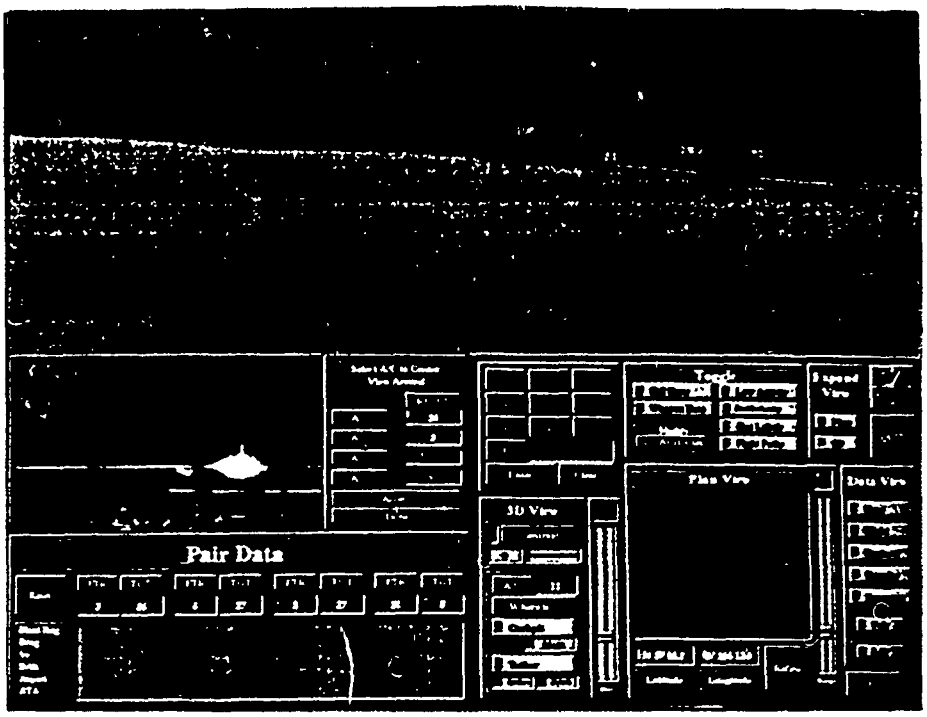

Figure 64. Pair Data

VI. RDT *Performance*

For each task there are numerous "yardsticks" by which the success of that task can be measured. For *RDT*, the ultimate measure of its success would be the incorporation of the software and hardware into an operational environment where it would be used as the debriefing tool it was designed to be. Other "yardsticks," or metrics, exist however, which can also lend insight into how well the software could be expected to perform in that operational world. This chapter measures *RDT* performance in three areas: data translation, visualization, and user task completion.

6.1 Data Translation

In order to establish the context under which data translation performance tests were completed, some background information is required. This section first describes the contents of the data file that was used for Data Translation Component (DTC) evaluation and then explores the issues of dead reckoning and speed for both single and multi-processor machines.

6.1.1 Red Flag Mission Description. The Red Flag exercise data used for DTC testing was recorded during the morning mission on 15 Jul 93. A total of 53 aircraft participated during the mission, and of that total 7 different aircraft types were represented. The numbers and types for the mission are listed in Table 9.

Aircraft Type	Number
F-4	2
F-15	16
F-16	27
B-52	2
C-130	2
KC-135	3
E-3A	1
Total	53

Table 9. Aircraft Used in a Typical Red Flag Mission

Not included in the table are the numbers and types of simulated weapons which were fired at the Red Flag participants. Approximately 125 weapons entities were inserted and removed from the exercise as a result of fire and detonation events generated by the RFMDS CCS.

Missions flown during Red Flag exercises have many elements in common and include the following phases:

1. Air Refueling/Marshalling - Aircraft join up with their designated tankers to top of their fuels tanks and assemble the strike formations.
2. Push - Arrive at the start route point in the strike formation. Aircraft are armed and ready for combat.
3. Low Level Ingress - The strike formation, or package, uses the terrain to avoid detection by enemy air defenses.
4. Aerial engagement - Enemy air forces detect and engage the strike package in air-to-air combat.
5. Air-to-Ground Weapon Employment–Strike package locates their assigned targets and commence weapon deliveries.
6. Defensive Maneuvering–Enemy positions engage strike aircraft forcing the strikers to perform defensive maneuvers to stay alive.
7. Low Level Egress–Individual flights of strike aircraft return to friendly territory.
8. Return to Base–Post strike refueling if required and cruise back to base.

To help visualize the maneuvering that occurs in one or more of these phases, examine the air-combat engagement profile in Figure 65. This graph represents the flight path of an F-16 as is progressed from a patrol/search phase in Segments A,B to the commit phase in Segment C and finally through the engagement phases of Segments D,E. Maximum performance maneuvering most likely occurred in Segment E.

The mission flown on 15 July included all of these phases and is therefore representative of typical Red Flag missions. The 15 July data includes not only the cruise,

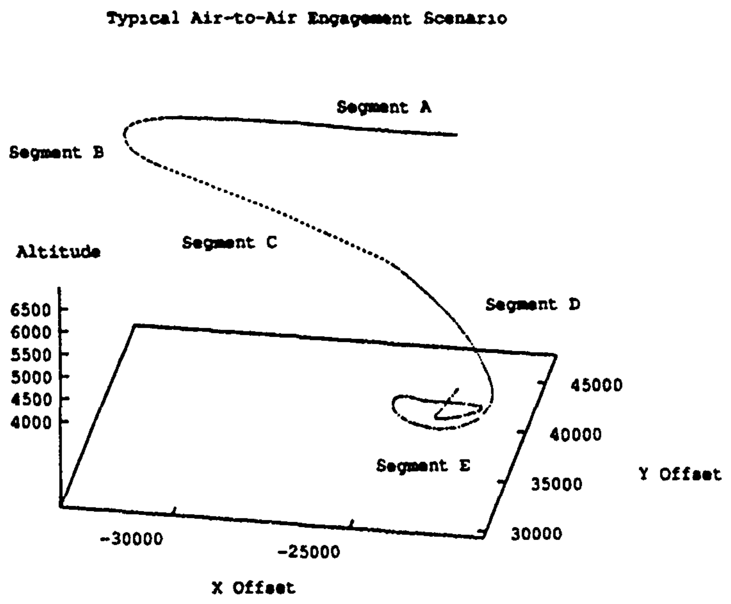

Figure 65. Air Combat Maneuvering Engagement Profile

or non-maneuvering, portions of the mission but also the air-combat and defensive maneuvering which tests a pilot's ability to maneuver his aircraft at the edge of its defined operating envelope. This broad range of maneuvering provides an excellent source from which assertions about the effectiveness of dead reckoning algorithms can be validated. Before examining the processing cost savings achievable with dead reckoning, the following section quantifies the information contained within the 15 July data file.

6.1.2 Data Description. The raw data file, converted from magnetic tape, contains 184 Megabytes(Mb) of information. About 30% of that file contained unused data and was trimmed down by the *XFilter* program to 123Mb. The total duration of the recorded data spans 2679 seconds, or 44 minutes and 39 seconds. The 123Mb of filtered data contained 25,775 RFMDS message blocks which must be parsed and translated by *Readred.* The breakdown by message type is shown in Table 10. The asterisked items in the table do not reflect the number of individual RFMDS messages in the file but rather the number of DIS PDUs that are generated as a result of parsing the contents of the respective messages. (The maximum number of individual messages that can be contained

Message Block Type	Name	Number
0	Unknown	8
1	Mission Data	11
2	GCI Mission Data	10
3	Maneuver Data	*788,452
4	Range Time	25,775
6	High-Activity Participant	9
7	GCI Data	25,564
8	Threat Mission Data	10
13	Range Status	25,753
15	Weapons Data	*8,730
16	Bomb Data	4
18	UHF Radio Frequency	8
20	Threat Data	25,563
21	Low-Activity Aircraft Data	1,558
22	Low-Activity Participant Data	11
23	Zone Entry/Exit	154
24	Bomb Target Data	9
29	Target Status	25,754

Table 10. Numbers of Individual RFMDS Messages within the 15 Jul Data

within the file for any single message type may not exceed 25,775 because only 25,775 message blocks were transmitted)

6.1.3 Dead Reckoning Cost Savings. The benefits of dead reckoning can best be seen from the data in Table 11. This table itemizes the number of Entity State, Fire and Detonation PDUs that are generated by *Readred* as it translated the 15 July data. The first column shows the number of PDUs sent when the position dead reckoning threshold is set to 1.0 meter. Column two is categorized similarly except that the dead reckoning threshold is set to 10.0 meters. This is the normal threshold used by *Readred.* Column three reports the numbers of PDUs transmitted when dead reckoning is disabled and forms the baseline for comparing the numbers of PDUs sent with and without dead reckoning algorithms.

The network message traffic reductions provided by the dead reckoning algorithms of the DIS standard are dramatic! By using a nominal 10 meter threshold and the 5

Type	1 meter Threshold	10 meter Threshold	No Dead Reckoning
ES PDUs Transmitted	177,277	44,919	788,452
Weapons PDUs Transmitted	6,801	1,677	8,730
ES PDUs Suppressed	613,127	753,730	0
Total PDUs	797,205	800,326	797,182
% Traffic Reduction	76.9%	94.2%	0%
Ave PDUs/sec	68	17	297

Table 11. Network Traffic Reductions with Dead Reckoning

second update rule imposed by the DIS draft standard, more than 94% of the PDUs that potentially would have been generated without the dead reckoning algorithms in place can be discarded. The narrower 1 meter threshold causes more PDUs to be transmitted but still allows 76.9% of the potential PDUs to be discarded. These findings are similar to those reported by Harvey(7:128).

The average number of PDUs broadcast per second shown in the last row of the table is a rough measure of the network broadcast capabilities required to transmit the total number of PDUs over the entire exercise time period. Naturally, some mission phases will exhibit higher PDU transmission rates than others and so the average rates in the table may be exceeded by a considerable margin.

Figure 66 shows a comparison between the flight path of the F-16 aircraft in Segment B described previously and the positions predicted by the dead reckoning algorithms. The dashed line represents the dead reckoned position of the aircraft and the periodic flight path corrections required when the time or position thresholds were exceeded. From a distance, the difference between the two flight paths is negligible, but examination of the dead reckoned flight path reveals a jagged, unrealistic path that could never be flown. Because of the periodic corrections required by the DIS standard, the dead reckoned flight path represents a suitable approximation for most applications. More restrictive thresholds produce closer approximations to the real flight path and fewer observable maneuvering irregularities but also incur additional network traffic penalties. These tradeoffs must be weighed for each DIS application.

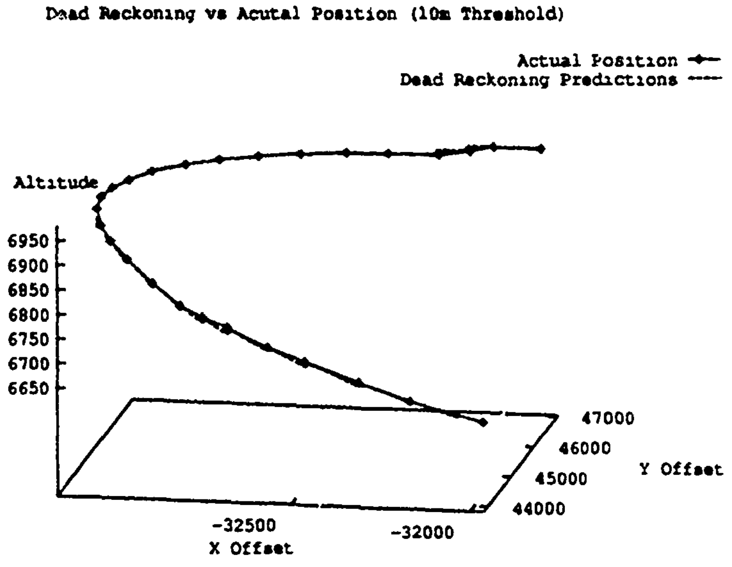

Figure 66. Dead Reckoning vs Actual Flight Data

6.1.4 Single vs Multi-Processor Machine. This subsection explores the processing capabilities of several single and multi-processor machines in the AFIT graphics lab to translate and broadcast the data in the 15 Jul data file. The small sample sizes of the test data gathered for Table 12 restrict the number of credible conclusions which can be drawn about processor capabilities. This data is included only as an initial indication of relative performance. External network traffic, operating system process context switching, and other background processing are all factors which could not be controlled during the test period.

Type	CPU (Mhz)	# of Processors	DR (1)	DR (10)	Turbo (1)	Turbo (10)	Turbo (NoDR)
Indigo	33	1			469	918	
IndigoEX	100	1	2,979	2,683	705	288	3,236
510/VGXT	50	1	2,985	3,152	815	274	3,164
440/VGXT	40	4	3,041	2,934	1,242	554	3,382
Onyx RE^2	100	4	2,682	2,681	961	332	4,482

Table 12. Relative Processor Performance

Columns 1,2,and 3 of Table 12 identify the SGI machines used for the test and a few of their characteristics. The columns labeled "DR" indicate that the dead reckoning algorithms were used during the translation process. The numbers in parentheses reflect the dead reckoning threshold, in meters, used for a given set of test runs. The label "Turbo" in columns 6, 7, and 8 means that the normal time synchronization mechanism used in *Readred* was disabled and that the program was allowed to translate and transmit the PDUs as fast as possible. If the dead reckoning algorithms are not used and the time synchronization mechanism is disabled, *Readred* generates a PDU for all participants every 100msec. In the case of the 15 July data, this equates to almost 800,000 PDUs.

What do the test results indicate? First, single processor machines, executing the same code as the multi-processor machines, can compete favorably with the multi-processor machines. In turbo mode, the Indigo EX consistently out performed the multi-processor machines and was comparable in the normal dead reckoning mode. Second, further refinement of the time synchronization mechanism is required in order to bring the replay time in line with the real elapsed time of 2679 seconds. In nearly all of the DR mode test cases, *Readred's* elapsed time exceeded the RFMDS elapsed time. Third, an uncontrolled test environment, such as the environment in which these performance gauges were conducted, can significantly skew results. Many entries in the table point to apparent, counter-intuitive irregularities. For example, how can the Indigo, a 33 Mhz machine, in turbo mode at the 1m threshold process the data faster than the 100 Mhz, 4 processor, Onyx, or why would the Onyx process data significantly slower than any of the other test machines when every PDU is being broadcast? Further testing in a controlled test environment is necessary before any other conclusions can be drawn.

(After the data was compiled for Table 12, a new version of the network daemons was installed and tested at AFIT. The preliminary results indicate that the transmission speeds can be **increased by a factor of four**. The 440/VGXT translated and broadcast the 15 July data in the turbo, no-dead-reckoning mode in a time of 799 seconds. This is a drastic improvement over the previous capabilities with the old network daemons. In the normal time synchronization mode the completion time was comparable to other results, 2825 seconds. The turbo (10) mode completion time for the 440/VGXT was 218 seconds.

These improved transmission capabilities allow file-based mission replay using *Readred* at speeds up to 12 times actual speed.)

Network reliability is another issue that bears mentioning. Quantitative data is not available from *RDT* to show how many PDUs are transmitted by *Readred* and not received by RDT_{dis}, but the situation does occur. This problem is an inherent characteristic of the underlying philosophy of DIS and its reliance upon the UDP/IP protocols where data receipt is not guaranteed. Under normal conditions with aircraft maneuvering through the exercise airspace, loss of a few Entity State PDUs is not even noticeable. However, when weapon events are added to the complexity of the simulation, the loss of a single detonation PDU is very noticeable. Instead of the weapon detonating at the required location, the weapon's location continues to be dead reckoned for an additional 12 seconds and the detonation results are not available. If a Fire PDU is not received by RDT_{dis}, the weapon is correctly propagated through the scene by the Entity State PDUs that are broadcast, but the shot cannot be validated because the range, bearing, and closing velocity calculations are not performed. This detracts from *RDT's* usefulness as a debriefing tool. Repeated demonstrations with the 15 July data file indicate that roughly one to five percent of the Fire and Detonation PDUs are lost. It is currently not possible to determine the number of Entity State PDUs which may be lost because of network difficulties.

6.2 *Visualization*

A performance measure that has often been used to compare relative processor capabilities is the number of frames per second that the graphics engine can generate. This measure can only be used for machine comparisons if the geometry, shading, and lighting conditions are the identical on each computer. If identical conditions cannot be duplicated the frame rate comparison must be discounted. This section details the polygonal complexity incorporated into the Red Flag terrain model in an effort to provide some context for the frame rate performance statistics given later.

6.2.1 Red Flag Terrain Complexity. The Red Flag terrain used in RDT_{dis} is constructed with four levels-of-detail. At the lowest level-of-detail, each of the six 1°

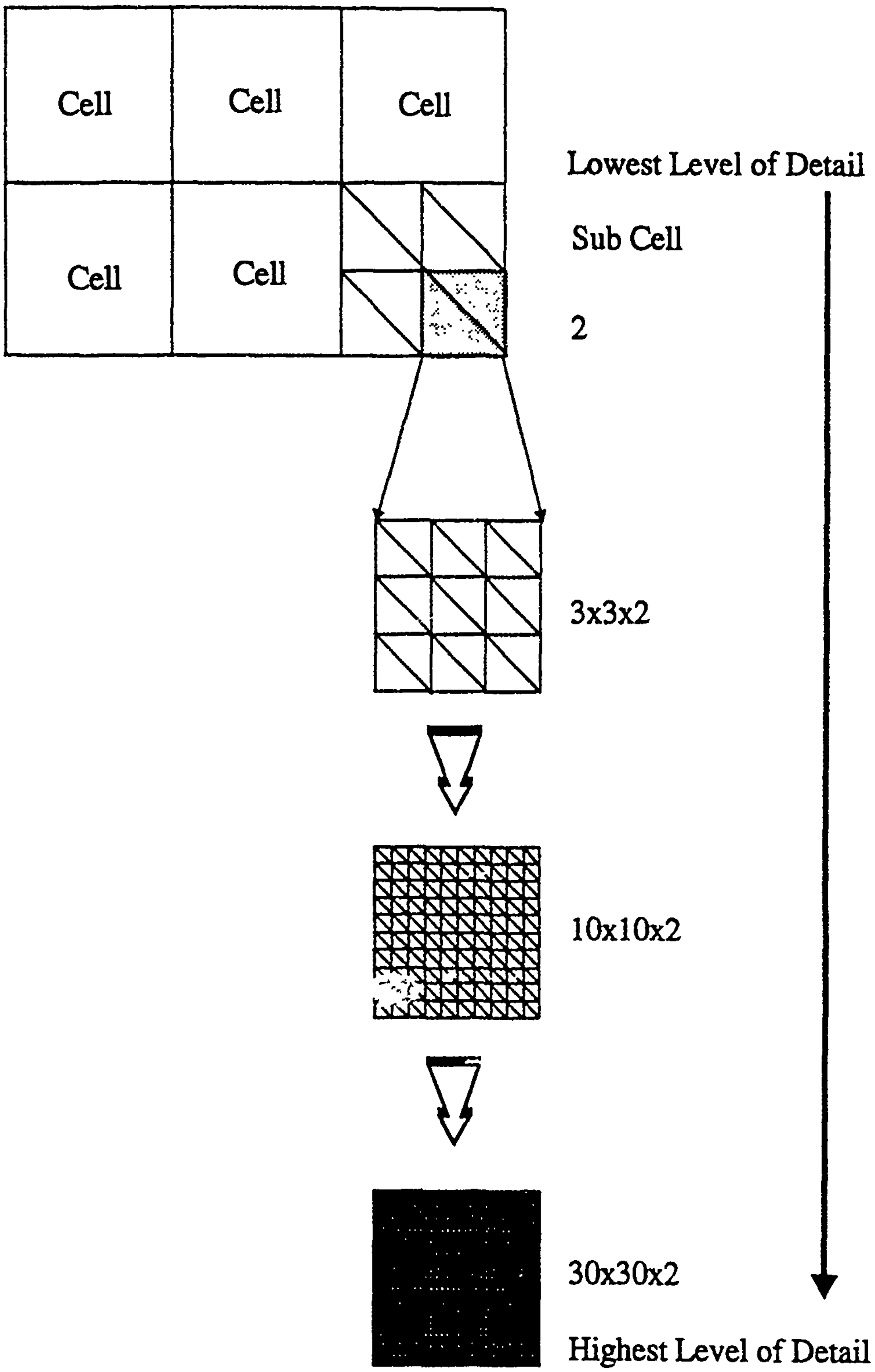

Figure 67. Red Flag Terrain Geometry

Level	# of Polygons
Lowest	48
Low	432
High	4800
Highest	43,200

Table 13. Polygonal Complexity in Red Flag Terrain

latitude x 1° longitude cells is divided into four equal subcells, each containing two triangles. This gives a total of 8 polygons per cell and 48 polygons for all 6 cells. Figure 67 shows how each of the subcells is further subdivided at the next higher level-of-detail into 18 triangles. A further subdivision of the subcell creates a level-of-detail that contains 200 polygons and at the highest level-of-detail the total number of polygons for each subcell is 1800. Table 13 gives the total number of polygons contained at each level-of-detail for the Red Flag terrain. The subdivision of each of the cells into four separate subcells was done to allow the *Performer* software an opportunity to cull, or clip, pieces of geometry from the scene and to provide finer control for the level-of-detail switching. Level-of-detail switching and culling reduce the total number of polygons visible in the scene at any one time and directly contribute to higher frame rates.

6.2.2 IVC Frame Rates. Without the ability to poll the *Performer* software for the number of polygons drawn in a single frame, it is extremely difficult to determine the polygonal scene complexity and make rigorous frame rate comparisons. The frame rate statistics provided in Table 14 are therefore presented in order to lay an intuitive foundation for achievable frame rates on the SGI machines for RDT_{dis}. The statistics, taken from the output of the pfDrawChanStats procedure, were averaged over an extended period of time and reflect the following general conditions:

1. Approximately 42 aircraft models
2. An average of 6 weapon models
3. Red Flag terrain with four texture-mapped levels-of-detail and multiple cells visible

Machine Type	Performer Version	Normal (2 Chan)	Expanded (1 Chan)	Full Screen (1 Chan)
440/VGXT	1.1	3-6	4-7	3-6
Onyx RE^2	1.1	10-15	15-20	10-15
Onyx RE^2	1.2	12-20	15-30	15-20

Table 14. Frame Rate Performance

Included within the frame rate statistics are a number of other graphics activities. These include the labeling of each aircraft with a number or call sign, drawing an optional flight-path history trail, drawing the HUD with a stroked font library, managing *Forms* window events, and updating the *Forms* based data views. These additional activities can reduce overall frame rates anywhere from 2-4 frames per second. The low frame rates shown for the 440/VGXT are principally a result of the texture mapping used for the terrain. Other tests conducted on the 440/VGXT without texture mapped terrain generally yielded frame rates that were only 2-4 frames per second slower than the Onyx RE^2 machines using *Performer* 1.1. The optimum configuration at AFIT for RDT_{dis} is the four-processor Onyx RE^2 workstation running *Performer* version 1.2. This configuration meets and exceeds the desired frame rates for the *RDT*.

6.3 User Evaluation

To determine *RDT's* suitability for performing mission monitoring and debriefing tasks the interface requirements defined in section 4.1.1 can be used as a partial measure of merit. Captain Jim Raulerson, an experienced mission analyst from the 57TG, was asked to become acquainted with the RDT_{dis} interface and then use RDT_{dis} to evaluate the performance of aircrews participating in the 15 July mission. At the end of the training and evaluation period the analyst was asked to subjectively rate $RDT'_{dis}s$ capabilities in comparison to RFMDS and to mission needs. Table 15 gives the results of that evaluation. Additional capabilities, not included in section 4.1.1, surfaced during the evaluation and are included in the table as well. Two things must be kept in mind, however, when examining the table–the scope of the *RDT* project was very limited compared to the full capabilities of RFMDS and the design of an equivalent system was never intended. The goal of *RDT* was to demonstrate the feasibility of developing a remote debriefing tool that used the DIS protocol.

Column one of Table 15 identifies a list of specific capabilities that are possible on one or both of the systems. Check marks in column two indicate which capabilities can be found on RFMDS. A similar mark in column three indicates the capabilities included in RDT_{dis}. When both systems include a capability and one system offers a significantly higher capacity a '$\surd^{+}$ symbol is used.

Examination of the table reveals that there are significant capabilities which RFMDS possesses that still need to be incorporated into *RDT*. The two most important capabilities are communications and replay. The ability to monitor the radio communications is crucial in evaluating the situational awareness of the aircrews, flight coordination, and weapon employment. (This last area is particularly vital because some aircraft do not send fire control data to the RFMDS and thus no weapon simulations are computed. In addition, the voice shot calls can serve as an alternate source of weapon fire events.) A variable-speed replay capability is the backbone of the debriefs which follow the mission. *RDT's* limited replay capability available by recording the mission data with *Convert* and then using the data file with *Readred* provides a mission replay capability, but does not allow the mission to be paused or the replay speed to interactively be changed.

RDT's strengths include realistic aircraft and terrain representations, extensive capabilities to modify view orientations with both spaceball and mouse, simple user interface, analysis data, and most importantly, DIS compatibility.

A table of capabilities, like Table 15, provides some useful insight in determining $RDT'_{dis}s$ adequacy for debriefing a live mission. But the important question which must be answered is 'can RDT_{dis} provide the same analysis capabilities that RFMDS provides for live air-combat-maneuvering?' The answer from Capt Raulerson was a resounding "yes."

Item	RFMDS	RDT_{dis}
Plan View with Pan/Zoom Capability	$\surd$	$\surd$
Centroid View + Orientation modifier	$\surd$	$\surd^{+}$
Pilot View + Orientation Modifier	$\surd$	$\surd^{+}$
Tether View + Orientation Modifier		$\surd$
Ground Threat View	$\surd$	$\surd$
Flight Data Display	$\surd$	$\surd$
Engineering Data Display	$\surd$	$\surd$
Exercise Data Display	$\surd$	$\surd$
Weapon Summary Data Display	$\surd^{+}$	$\surd$
Archive Summary Data	$\surd$	$\surd$
Summary Data Reports	$\surd$	
Simultaneous Data, Plan and 3D Views	$\surd$	$\surd$
Aircraft Pairing Information	$\surd$	$\surd$
Radar and IR Missile Lock Indications	$\surd^{+}$	$\surd$
Aircraft Flight Path History Trails	$\surd$	$\surd$
Hard Copy Printouts	$\surd$	
Radar and IR Missile Seeker Position Symbols in Pilot View	$\surd$	
Restricted Airspace Depiction	$\surd$	$\surd$
Modify Aircraft Color	$\surd$	$\surd$
Monitor 8 Radio Channels	$\surd$	
Variable Speed/Pause Replay	$\surd$	
Weapon Employment from Console	$\surd$	
Air-to-Ground Weapon Scoring	$\surd$	
Threat Operator Video	$\surd$	
Geographic Bullseye Reference Point		$\surd$
Aircraft Pairing with Bullseye Point		$\surd$
Realistic, 3D Filled Polygon Models		$\surd$
Elevation Shaded Solid Terrain		$\surd$
Add Models for Threats and Targets		$\surd$
Aircraft Locator		$\surd$
Detach and Fly through Scene		$\surd$
Establish Viewpoint from *ANY* position		$\surd$
Weapon Detonation Animation Effects		$\surd$
Portable		$\surd$
DIS Compatible		$\surd$
View DIS Entities from other Sites		$\surd$
Broadcast DIS data to *unlimited* # of Sites		$\surd$
DIS Simulator Interaction with Live Aircraft		$\surd$

Table 15. RFMDS/RDT_{dis} Capabilities Comparison

VII. Conclusions and Recommendations

This chapter summarizes the success of the *RDT* as measured against the original thesis statement given in Chapter I and then makes recommendations for future effort and research. The concluding portion of this chapter reports on capabilities, derived from this thesis effort, which if further developed may significantly improve the training of today's aviators and better prepare them to meet the challenges of the future.

7.1 Thesis Statement Revisited

The original thesis statement, or goal, can be subdivided into a number of supporting objectives. *RDT* can be evaluated against each of these supporting objectives to determine the overall success of the thesis project.

Objective number one was the development of a hardware and software system which proved the feasibility of a DIS-based Red Flag mission monitoring and debriefing tool. Successful achievement of this objective was demonstrated during on-site testing at Nellis AFB. By simultaneously using one of the RFMDS DDS consoles and *RDT*, the images of both systems were projected onto two large viewing panels in one of the debriefing rooms at Red Flag. Both panels showed the live activities of a two-ship as it conducted test flights on the Nellis ranges. The source for the image on one panel was RFMDS and the source for the other image was *RDT*. Comparison of the images on the two panels showed identical aircraft movements by both systems. Flight maneuvering in the RFMDS panel was mirrored in the *RDT* panel. As the flight crossed one of the terrain features drawn by RFMDS, *RDT* depicted an identical movement over the same terrain feature. The ability to monitor live aircraft on the Nellis ranges using *RDT* is now a reality.

Objective number two specified an ability to transmit the aircraft telemetry to remote sites. This capability was also simulated during the on-site test at Nellis by transmitting the DIS PDUs over a fiber optic cable from the Data Translation Component (DTC) in room 116 to the Interface and Visualization Component (IVC) located in room 234. Although the distance is likely under 150ft, it demonstrated the capability for the DIS data to be transmitted and successfully received using long distance mediums. The set of fiber modems could just as easily have been an equivalent set of hardware that could convert the Ethernet protocols to those used with T-1 lines or other long distance networks.

Preparations are underway for a final demonstration of *RDT's* capabilities to transmit and receive DIS data over long distance networks. During this demonstration, Nellis, AFIT and the ARPA Simulation Center will be connected on a wide area network. Live Red Flag mission data will be broadcast to AFIT and ARPA. Both sites will monitor the live exercise by using *RDT*. In addition, the Virtual Cockpit simulator at AFIT will receive the DIS data and be able to "fly in the exercise." The Virtual Cockpit will have complete visiblity of all of the Red Flag aircraft and will be able to acquire the aircraft not only visually but also on its simulated radar, perhaps even employ weapons. Activities of the Virtual Cockpit will be seen at both ARPA and Nellis on *RDT*. This will be the beginning of simulator interaction with live aircraft.

Objective number three required an interactive interface to a state-of-the-art three-dimensional image generator. The *Forms* software contributed the tools necessary to create a graphical user interface which incorporated many of the same capabilities for view selection and control that exist on the RFMDS DDS consoles. The Silicon Graphics Onyx machines with their Reality Engines formed the backbone of RDT_{dis} and were directly responsible for the system's excellent frame rates. User evaluations confirmed equivalent capabilities for live monitoring of Red Flag missions.

The final objective stated that *RDT* be constructed with off-the-shelf network communications hardware and thesis software. All hardware used for the *RDT* implementation was purchased directly from commercial vendors and required no further modifications. Thesis software was tailored to the hardware components to create a complete system which fulfilled all of the objectives of this thesis project.

7.2 Future Work

The future appears bright for the world of distributed simulation. Among the top seven priorities that ARPA has chosen for future military technological research is the field of distributed simulation. DIS compatible applications, such as *RDT*, will benefit from the additional resources and efforts dedicated to distributed simulation environments. In the near term, a significant effort still remains to improve and extend the capabilities of *RDT*. The following list suggests a few of the areas where additional research is warranted.

1. Communications – The DIS standard can be employed to transmit and receive digitized voice data. Hardware and software packages exist on the market today which might be adapted for use with *RDT*. The ability to monitor radio communications would be a significant step forward for the remote debriefing concept.

2. Mission Replay – Additional effort should be dedicated to the development of a DIS data logger. The logger would function not only in a record mode, but also would be used to replay a mission, at variable speeds, from stored DIS PDUs. One of the challenges to be overcome here is the coordination with the Object Manager so that appropriate dead reckoning positions are calculated for the variable replay speeds.

3. Orientation Dead Reckoning – Smoother aircraft rotations about its three primary axes can be achieved by incorporating orientation dead reckoning into both *Readred* and the Object Manager. This will create a higher fidelity simulation and cause fewer distractions for the user.

4. Reengineering – Improve the current software architecture so that it can dynamically grow to meet the challenges of thousands of DIS entities interacting over a common network. The current limit is 500 players. This capacity has already been exceeded. Effort could also be expended to improve the user interface and possibly even incorporate voice technology to control RDT_{dis} functions.

5. Air-to-Ground Capabilities – Expand the current scope of *RDT* from air-to-air to include air-to-ground. This involves the addition of target entities onto the terrain and a means for target scoring. Data is already available within the RFMDS message blocks that identifies the location of bombs at the point of impact. Additional animation effects can also be incorporated to add a sense of realism.

6. Reports – No hard copy printouts are currently available with *RDT*. A capability to generate printouts of the aircraft flight paths during a mission would serve as a useful tool for debriefing. Weapon summary reports should also be included.

7. HUD Symbology – Develop a suitable method that can be used to provide IR missile seeker and radar target designator symbology onto the cockpit view's HUD. This is not a trivial matter. The azimuth and elevation angles for both the radar and the IR missiles is given in every maneuver data message. There is no mapping of these angles directly into any of the DIS PDUs. The articulated parts fields of the Entity State PDU have been considered; however, the use of dead reckoning algorithms to suppress the number of PDUs being broadcast complicates this issue. How do you dead reckon the position of the changing HUD symbology between PDU updates? Additional work that might be done in conjunction with efforts involving the Virtual Cockpit would be converting the generic HUD now used by RDT_{dis} to the new Air Force standard HUD symbology design.

8. Implement New DIS PDUs – Many of the engineering and flight data views contain fields that are currently not being filled because the data is not available in the DIS PDU. This difficulty can be overcome by implementing an *RDT* PDU that would contain such items as Indicated Airspeed, Mach Number, Angle of side slip, Crab Angle. A new question arises, however, "how often should these PDUs be sent and what effect will they have on the overall network traffic?"

9. Evaluate Secure Network Communications – What impact will there be on the system if the data is encrypted and decrypted. Can *RDT* be modified to work in a secure environment?

10. Consolidate CCC and DTC – Figure 68 illustrates how the hardware design of the current CCC and DTC might be reengineered to facilitate the consolidation of both the CCC and DTC into a single component. The PC and the SGI could, theoretically, be replaced with a single, high-speed, multi-processing workstation that possessed both Ethernet and RS-422 capabilities. This consolidation would reduce the number of serial devices in the data pipeline that are subject to failure and improve the maintainability of the system.

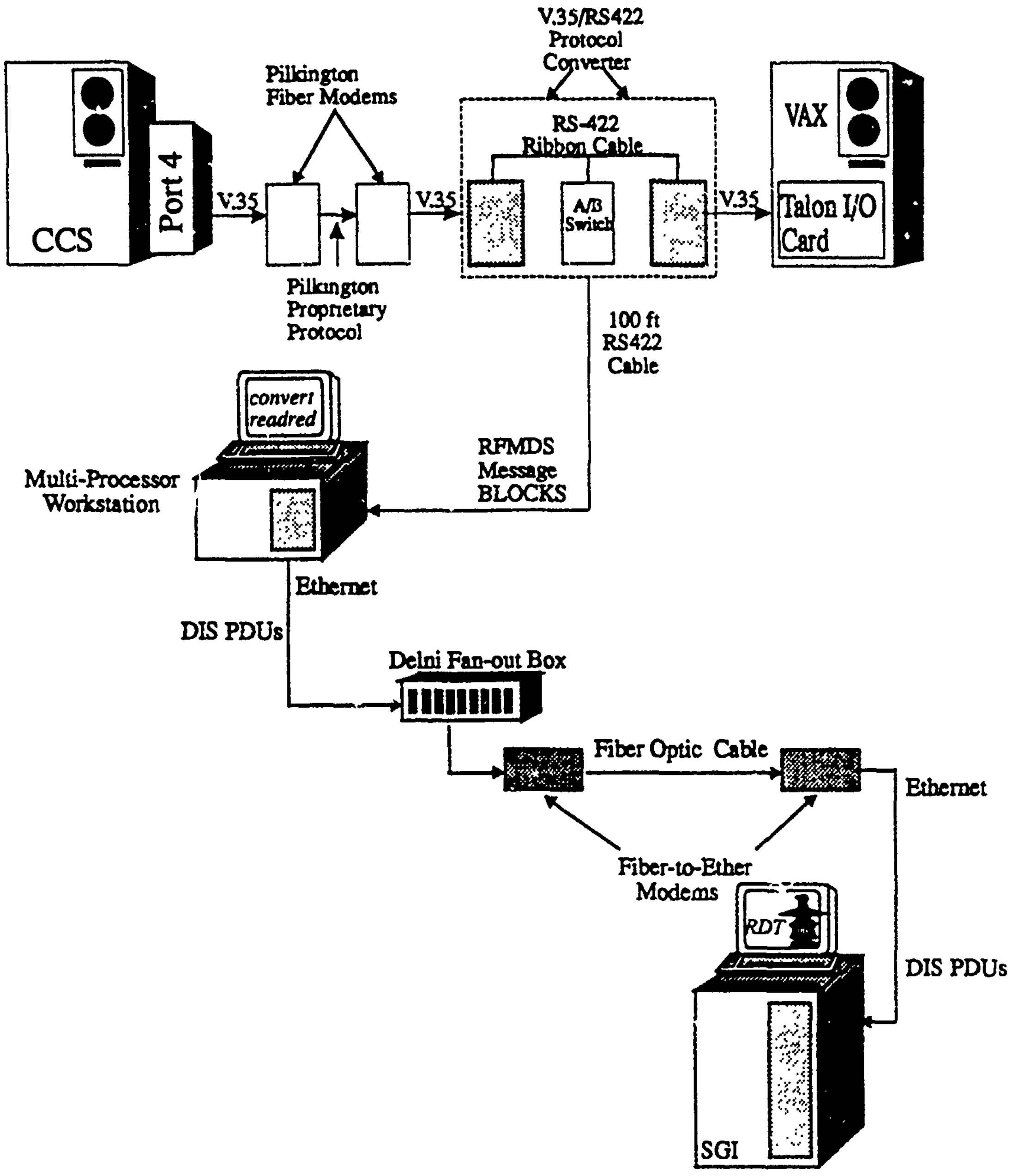

Figure 68. Consolidated CCC and DTC Design

7.3 Looking Ahead

Thus far this thesis has documented the design, implementation, and performance characteristics of the *RDT* yet it failed to answer the one question that champions a project's survival and future utility–'how can this new technology be applied to support the Air Force mission–**to Fly and Fight**?' This section addresses that question directly by looking first at *RDT's* current capabilities and then looking forward to capabilities that can be developed by extending this technology.

7.3.1 Current Capabilities. To explore what is possible today, the assumption must be made that funds would somehow be available to support the purchase of SGI workstations and dedicated, high-speed communications lines. Given that assumption, the following list details immediate applications for *RDT*.

- Remote mission analysis for 57TG, 422TES. Fiber optic cables scheduled for installation in the near future at Nellis AFB could serve as the communications backbone to the Red Flag building. *RDT* could be used to monitor test missions from their local sites allowing greater visibility to supervisors and test personnel.
- Aircrew debriefing at Red Flag. The thrust of *RDT* has always been to provide an air-to-air debriefing tool with capabilities superior to any normally available to a squadron. *RDT* can be used immediately at Red Flag as another resource to help debriefing process.
- Red Flag Familiarization Training. With additional SGI workstations and Virtual Cockpit software developed at AFIT, inexperienced pilots could sit down at a rudimentary flight simulator workstation and fly in a Red Flag exercise vicariously. Because the Virtual Cockpit is another DIS application, telemetry information being passed in the DIS PDUs is visible to the Virtual Cockpit. A young pilot could get a feeling for the intensity of a Red Flag mission and some of the terrain before ever strapping on a jet and putting his life on the line over the Nevada desert.

- Planning Staff Feedback. SGI workstations running *RDT* at exercise planning staff headquarters could provide immediate feedback to mission planners. Lessons learned on-the-spot could be incorporated quickly into follow-on mission plans.

- Remote mission analysis for 99BW. Installation or lease of T-1 communication lines between Nellis AFB and Ellsworth AFB, South Dakota would allow test missions flown by the 99BW on the Nellis ranges to be monitored. With the addition of audio and DIS replay capabilities, mission analysis could be conducted at Ellsworth thus reducing the need to deploy numerous individuals to Nellis for the mission test.

7.3.2 Future Capabilities. Looking a little further down the road to the time when squadrons might have access to SGI workstations running *RDT*, Red Flag deployment preparation would be superior to the local preparation now available. Squadrons could begin monitoring the live Red Flag missions well in advance and learn first-hand what tactics were working successfully. This cross-flow of tactical information could then be used to improve the squadron's tactics. Members of the squadron not participating in the deployment could monitor the squadron's success at Nellis and learn from their squadron mate's successes and failures. Virtual Cockpit software would also allow them to participate as 'phantom wingmen' in the exercise.

By developing an independent "black box" containing a GPS receiver and sufficient data storage, a simple device could be created which could be taken aboard any aircraft, or vehicle. The device would record the aircraft's position and orientation. Once the aircrews returned, the device could be taken to the squadron and the data downloaded into the SGI workstation for an immediate mission replay capability. The format of the data would be in accordance with the DIS standard thus allowing information exchange across either local or wide area networks.

Such a device could drastically improve mission debriefs in both the training and operational worlds. A complete mission analysis could be constructed quickly from the *RDT* displays and because the data could be archived, a complete set of scenarios could be constructed from live missions. These scenarios would show a trainee what the view out-the-window is supposed to look like. "Classic" engagements could be analyzed over and over from any number of viewpoints. Virtual Cockpit software would allow additional interaction with the stored scenarios and provide an opportunity to explore the 'what if' situations. No longer would the air-to-air instructor be tied to using hands, sticks, models or other training aids to convey the salient points of air combat maneuvering. The student could be **shown** the correct maneuver as well as ways to avoid common mistakes.

Missions flown against other units with the GPS based devices would allow all participants to view the mission simultaneously from remote locations by broadcasting their DIS data onto the network during the mission debrief. The visual presentations of the *RDT* displays and simple teleconferencing between the units could create a mission debriefing session rivaling RFMDS.

If a transmitter could be incorporated into the device, a real-time monitoring device could then be available in every squadron. The need for complex, instrumented ranges might be drastically reduced. Everywhere a squadron flies would become an instrumented range. Squadron supervisors could know the exact location of all of their aircraft. Aircrews, using *RDT*, could increase their awareness of squadron tactics and standards and improve their overall performance. A student in Undergraduate Pilot Training (UPT) could be shown that the barrel roll he thought he flew was nothing more than a corkscrew and then he could be given the opportunity to explore ways to improve the next time. Making information available to the aircrews about their flight performance allows problem areas to be identified early and firmly implants successful strategies in the aviator's mind.

The near future will also bring a capability for simulators to interact fully with live aircraft. As the real and virtual worlds grow closer together, the need for a debriefing tool that unites the activities of both worlds into a cohesive analysis will only intensify. Large-scale exercises involving thousands of DIS entities, both live and simulated, are on the horizon, and tools, such as *RDT*, which can provide the "big picture" to commanders, planners, and aircrews alike will serve a vital role in improving tomorrow's capabilities.

Appendix A. Command State Dynamic Models

This appendix contains the state transition diagrams for each of the command states of the user interface design described in section 4.1.2.2.

Figure 69 describes the states and the paths between them that are necessary to select an aircraft with any of the aircraft select buttons labeled A/C. The region within the dotted lines of Figure 69 is used in subsequent diagrams as a "black box" component which determines whether or not the number entered by the user corresponds to an active aircraft. No action is taken if the aircraft is inactive. If the aircraft number exceeds preset limits, an error message is displayed. If the number is validated, that aircraft number is made available to other functions within the IVC. This "black box" is used as a building block in the creation of complex activity paths.

Figure 70 illustrates the path that must be traversed in order to change the color of a given aircraft or change the length of all of the aircraft's flight path history trails. Figure 71 shows how a file is selected to store weapon events occurring during a given mission. Figures 72 and 73 portray how the data view control information is modified. Figure 73 also indicates the sequence of actions necessary to control the "Pair" data view display. Figure 74 shows how a single button press can be used to change the state of control panel objects from on to off or *vice versa*. Figure 75 depicts the paths used to change between the different 3D views and their subviews. Figure 76 illustrates how the control of the graphic's window is managed so that the views can be enlarged from their normal 1/4 screen size up to the use of the full screen. The "command state" needed to activate a larger positioner which provides a finer level of view movement is shown in Figure 77. Figure 78 portrays the path through states that must be taken in order to present information to a user about a predefined topic.

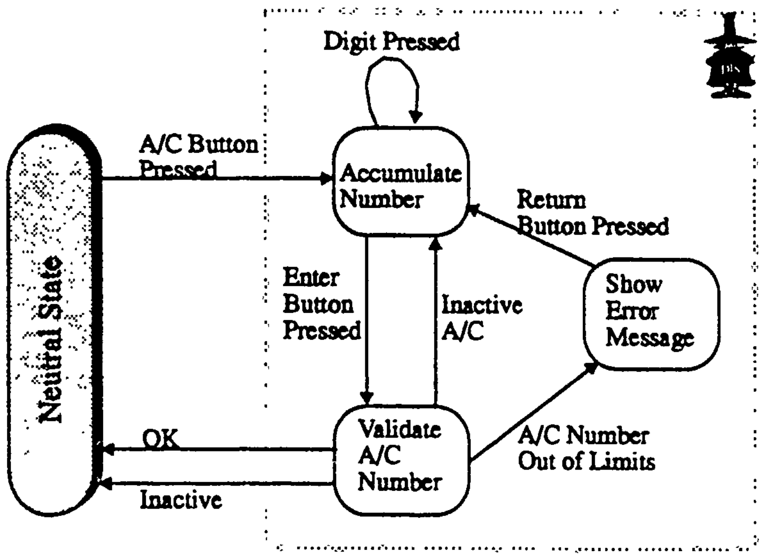

Figure 69. Validate Aircraft Command State

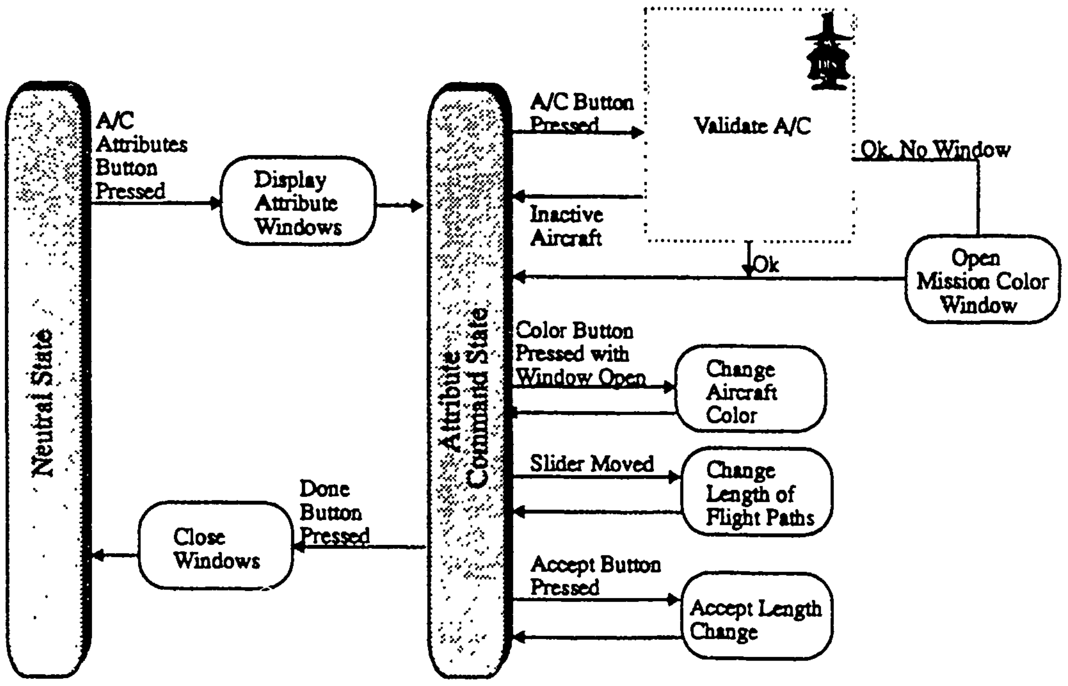

Figure 70. Attributes Command State

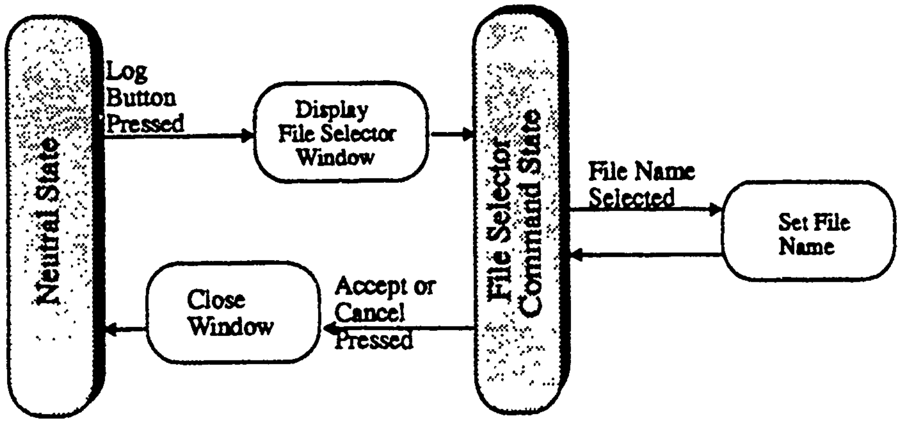

Figure 71. File Selector Command State

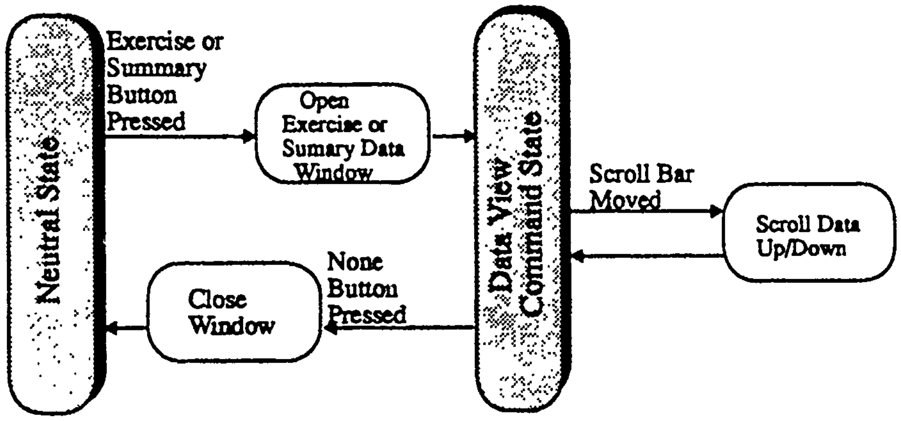

Figure 72. Data View Command State (Exercise and Summary)

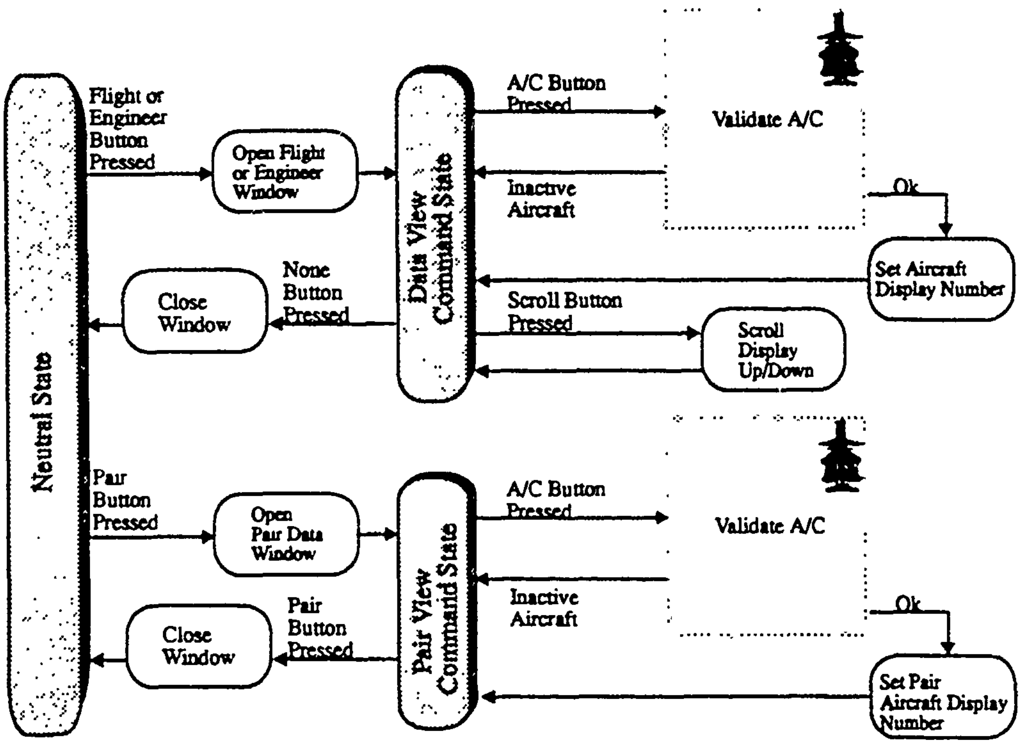

Figure 73. Data/Pair View Command State (Flight, Engineer, and Pair)

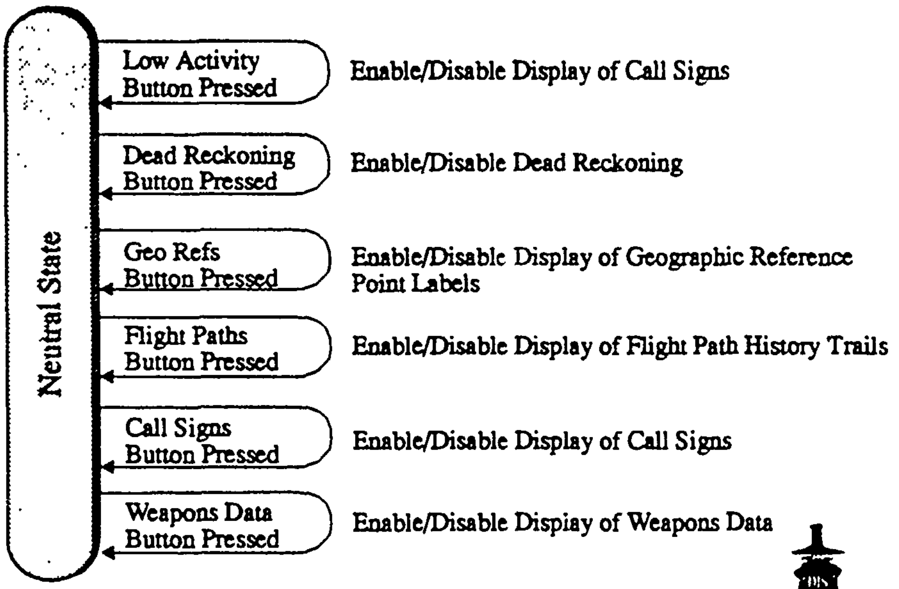
Neutral State
Low Activity Button Pressed
Enable/Disable Display of Call Signs
Dead Reckoning Button Pressed
Enable/Disable Dead Reckoning
Geo Refs Button Pressed
Enable/Disable Display of Geographic Reference Point Labels
Flight Paths Button Pressed
Enable/Disable Display of Flight Path History Trails
Call Signs Button Pressed
Enable/Disable Display of Call Signs
Weapons Data Button Pressed
Enable/Disable Dispiay of Weapons Data

Figure 74. Appearance Toggle Buttons

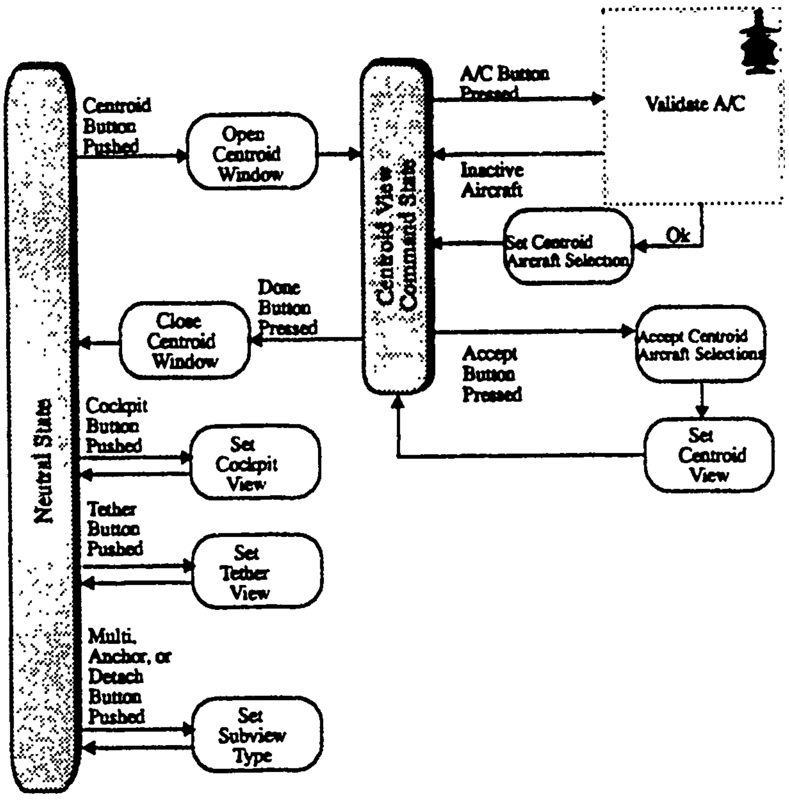

Figure 75. View Manager Command State

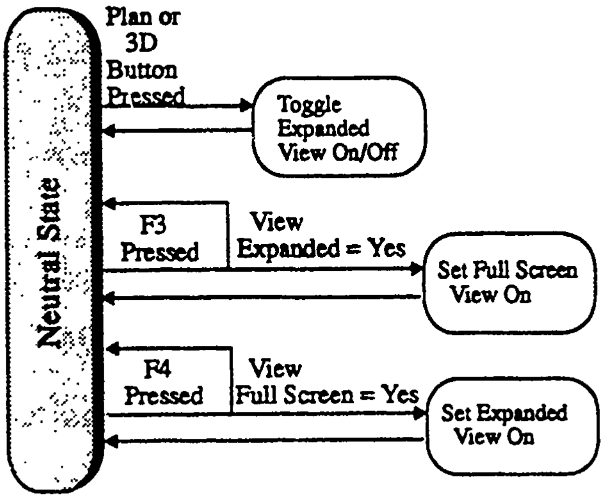

Figure 76. Expand View Command State

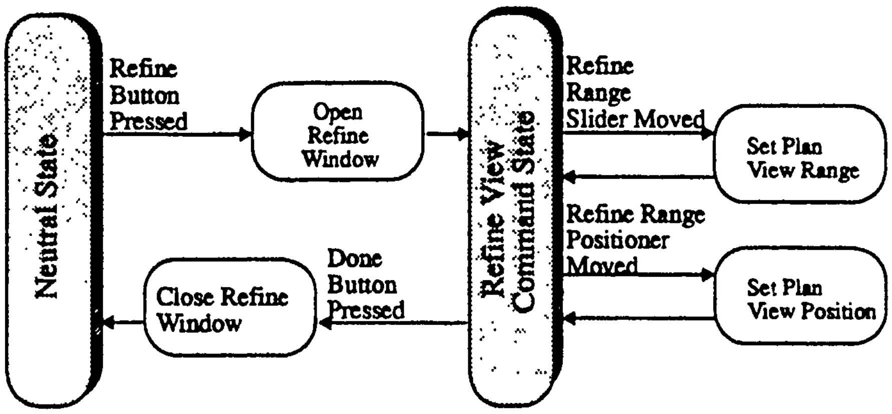

Figure 77. Review View Command State

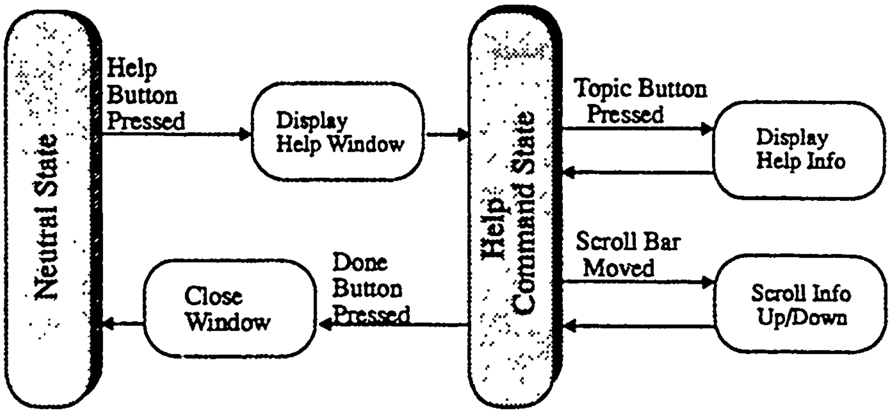

Figure 78. Help Command State

Appendix B. Convert

```
NAME
     Convert -- Convert RFMDS message blocks to Ethernet packets

SYNOPSIS
     convert [options]

DESCRIPTION
     Convert is a program which parses out the individual
messages contained within the RFMDS CCS-to-DDS message blocks.
The source of the block may be either a disk file or V.35 port.
The disk file must be a raw, unfiltered, CCS data file. Messages
parsed out of the blocks by Convert are output onto Ethernet port
1500.  Message output may optionally be saved to disk.  The
program terminates on keyboard input.

OPTIONS
     -b size
          Sets the read buffer to size bytes. Default size
          is 6144 bytes.

     -h host
          IP address of the machine that the messages are to be
          sent to.  For example, 129.92.101.111 -- Michelangelo
          at AFIT (default IP address).

     -r readfile
          Name of the raw data file that is to be used for the
          source of the RFMDS message blocks.

     -w writefile
          Name of the data file where the individual messages are
          to be archived for later use by Readred.

     -d
          Enable debug mode.

EXAMPLES
     convert -h 129.92.101.117 -r \RFMDSDAT\15JUL93.TLN

          Read message blocks from the 15JUL93.TLN data file and
          sends the individual messages to host 129.92.101.117 on
          port 1500.
```

```
convert

     Here the Convert program reads message blocks
     from the V.35 port and sends the individual messages
     out port 1500 to host 129.92.101.111.
```

HARDWARE REQUIREMENTS

Convert runs on an 80386/80486 Computer with processor clock speeds of at least 33Mhz. An Industrial Computer Source ACB5 V.35 interface card and a Western Digital 8003 Ethernet card are also required. Clarkson, public domain, packet drivers are used for the Western Digital Ethernet interface.

SOFTWARE SOURCE CODE

The source code for convert is archived at AFIT in /usr/people/wb/src/RDT/ccc/rdt_pc.zip on the Onyx machines. This compressed .zip file contains all of the source and libraries necessary to compile and build an executable program. The .zip file may be uncompressed by using the DOS PKUNZIP.EXE program . The source code also exists on the 486 computer in the Graphics Lab (Escher). The raw data file used by convert is located on Escher in the \RFMDSDAT directory and is called 15JUL93.TLN.

BUGS

The write option as currently implemented does not generate the correct data format. Planned correction of the problem is scheduled for Jan 94.

AUTHOR

Bruce Clay with minor modifications by Mike Gardner (1993)

Appendix C. Readred

NAME
    Readred -- Translate RFMDS messages into DIS PDUs

SYNOPSIS
    readred -d [network|datafile] [options]

DESCRIPTION
    Readred is the data translation component (DTC) of RDT. Readred accepts individual RFMDS messages from either the convert program as broadcast over an Ethernet network or from a disk file. Output from the program consists of Entity State, Fire and Detonation PDUs IAW the DIS v2.0.3 draft standard. When using the network as a source of input, both the receive and send daemons written by Bruce Clay must be installed on the host system prior to running Readred. Only the send daemon is required if the input source is from a disk file. (Ensure that the port being used by the convert program is the same number as the port being monitored by the receive daemon.) The Readred program terminates after an input data file has been completly read or the user interupts execution with a control-C.

The datafile used with Readred MUST have first been filtered by the XFilter program. Readred will not function correctly if raw RFMDS message blocks from a raw data file are used as input.

The redflag.cfg must be in the same directory as the executable Readred. Redflag.cfg contains reference information about the different message types used by RFMDS and the mappings between the redflag aircraft type identification numbers and the DIS Entity Type record. Mappings are present for both aircraft and weapons. The format definitions for the Redflag.cfg are included within the header information of the file.

All DIS PDUs generated by Readred use unique Entity ID record values. Until a permanent site ID is assigned to Nellis AFB, 711 is used to identify the site and 99 is used to identify the host. RDTdis looks for the 711/99 site/host values to differentiate a live Red Flag participant from any of the other DIS entities on the network. If these values change, corresponding changes in the RDT Object Manager must also be made.

Additional output generated when the input source is a data file consists of a tally of the number of individual RFMDS messages processed in the file except for message types 3 and 15. The tallys given for type 3, maneuver data, and type 15, weapons data, reflect the Entity State PDUs generated as a result of the RFMDS maneuvei and weapons data messages. Aircraft whose location data is known to be questionable will not have PDUs generated. A tally of the number of such occurances is also displayed with the statistics. The elapsed time display reflects the number of seconds used to process the file. The RFMDS elapsed time reflects the difference between the first RFMDS range time message in the file and the last range time message processed.

OPTIONS

-a

Analyze the network traffic for each of the high activity aircraft. The amounts to a PDU talley for each of the aircraft.

-b AircraftNumber

Restricts the broadcast of Entity State PDUs for all high-activity aircraft except AircraftNumber. This provides a way to isolate the activities of a single high-activity player.

-e

Show non-fatal error messages

-f

Use a socket based interface to control activities within Readred during execution. This has not been tested and is considered unreliable.

-i [lo|hi|wpn|fd|all]

Print (inspect) the contents of the specified PDUs as they are broadcast onto the network. (Debug option) Select between low-activity, high-activity, weapons (ES, FIRE and DETONATION), weapons (FIRE and DETONATION only), or all PDUs.

-m

Print the PDU message definitions taken from the redflag.cfg file. (Debug Option)

-n DeadReckoningThreshold

Specify a new position dead reckoning threshold tolerance. The default tolerance is 10.0 meters.

-o

Enable orientation dead reckoning. Not currently used.

-p

Print the contents of the participant records when they are received (Debug option)

-r

Disable dead reckoning. PDUs are generated for all aircraft at the RFMDS 10Hz rate. If dead reckoning is disabled inside Readred, dead reckoning inside RDTdis should also be disabled. This combination allows a direct input of the RFMDS data into RDTdis.

-s

Print (show) the RFMDS message types as they are received.

-t

Enable TURBO mode. PDU and RFMDS time synchronization is disabled. PDUs are broadcast as fast as possible. The -t and -r options can be used for network throughput testing since the maximum number of PDUs are generated with these options enabled. (Only valid for data file input. Network input is processed in real time.)

-v

Verbose mode. Show all debug statements. Use only as a last resort because of the volume of information that is generated.

-x AircraftNumber

Extracts the time and location for the specified AircraftNumber to the "xdata.log" file. These values are stored in ascii format thus allowing them to be used as input to programs such as GNUPLOT which can then analyze the location information generated by RFMDS.

-w

Print RFMDS weapons data. (Debug only)

-z

Show Zulu Time. (Debug Option)

Examples:

```
readred -d data/15Jul.filter -n 1.0
```

Start readred using the 15Jul.filter data file and use a 1.0 meter dead reckoning threshold tolerance.

```
readed -d network -s -p
```

Execute readred with input data coming from the network. Show the message types being processed and also print the participant data when received.

```
readred -d data/15Jul.filter -t -r
```

Disable dead reckoning and time synchronization to send PDUs as fast as the system will allow. This is the maximum throughput test mode.

```
readred -d network -x 24 -b 24
```

Entity State PDUs for aircraft number 24 are broadcast while all other Entity State PDUs for high-activity aircraft are suppressed. At the same time, the location data is saved into xdata.log for aircraft number 24.

SOFTWARE SOURCE CODE

The source code for Readred is located in the /usr/people/wb/src/RDT/dtc directory.

SEE ALSO

Convert, XFilter

AUTHOR

Bruce Clay
Major Michael Gardner

Appendix D. XFilter

NAME

    XFilter -- Filter RFMDS message blocks

SYNOPSIS

    xfilter [options]

DESCRIPTION

XFilter is a utility program that accepts as input a raw RFMDS data file, which most likely was downloaded from magnetic tape, and outputs a file containing consecutive individual RFMDS messages. RFMDS message blocks sent to the VAX computer at RECOM for post-mission analysis contain 6144 bytes. If the number and contents of the individual messages contained within the block does not fill the entire block the space remains unused. Over the course of a single mission approximately 1/3 of the space within the fixed size blocks is unused. The XFilter program skips any unused portion of the block as it parses out the individual messages. An additional filtering option is possible with the -f switch.

XFilter's other primary use is to extract portions of the raw data file into an output file by specifying the beginning block number and the number of desired blocks to be filtered. The input block size is 6144 bytes.

OPTIONS

-f

Filter out all RMFDS messages except the high and low participant data messages, maneuver data message and the low-activity data message. This switch effectively eliminates all weapons information in the new output file.

SOFTWARE SOURCE CODE

The source code for xfilter is in the /usr/people/wb/src/RDT/xutils directory. The only raw data file that exists at AFIT at present is in the /nipper3/mgardner directory and is called 15JUL93.TLN.

AUTHOR

Major Michael Gardner

Appendix E. User's Manual

Welcome to the Remote Debriefing Tool (*RDT*) for Red Flag Missions. This user's manual is designed to familiarize you with the basic operations of the RDT. The manual contains three sections: Getting Started, Running RDT_{dis}, and Performing Analysis Tasks.

E.1 Getting Started

This section details the hardware and software requirements for setting up the program. Also included are directions for setting up the data pipeline and configuring RDT_{dis}.

E.1.1 Required Hardware. The following equipment is required to create the complete hardware pipeline from port 4 of the Red Flag Measurement and Debriefing System (RFMDS) to the Interface and Visualization Component (IVC) workstation. Those items which have an asterisk are the minimum items necessary for an abbreviated pipeline–*Readred* using a data file and broadcasting to the IVC workstation.

Qty	Item
3	V.35/RS-422 Protocol Converters
1	RS-422 Ribbon Cable with 3 connectors
1	RS-422 A/B Switch
1	RS-422 Cable, 100 ft.
1	ACB5 V.35 I/O Board for PC
1	Short V.35 cable from converter to V.35 PC board
1	Western Digital 8003 Ethernet Board for PC
1	Delni Fan-Out Box*
4	Thickwire Ethernet Cables*
2	Fiber Modems (Fiber - Ethernet)
1	Fiber Optic Cable, 100 ft.
1	IBM PC Compatible 80486 with Large Capacity Disk Drive
1	SGI Workstation (Single- or Multi-Processor)*
1	SGI Onyx RE^2 Workstation*

Table 16. RDT Equipment List

(Note: Another possible configuration is to use only the Onyx workstation and have *Readred* and RDT_{dis} running on the same machine. This configuration, however, yields slower frame rates.)

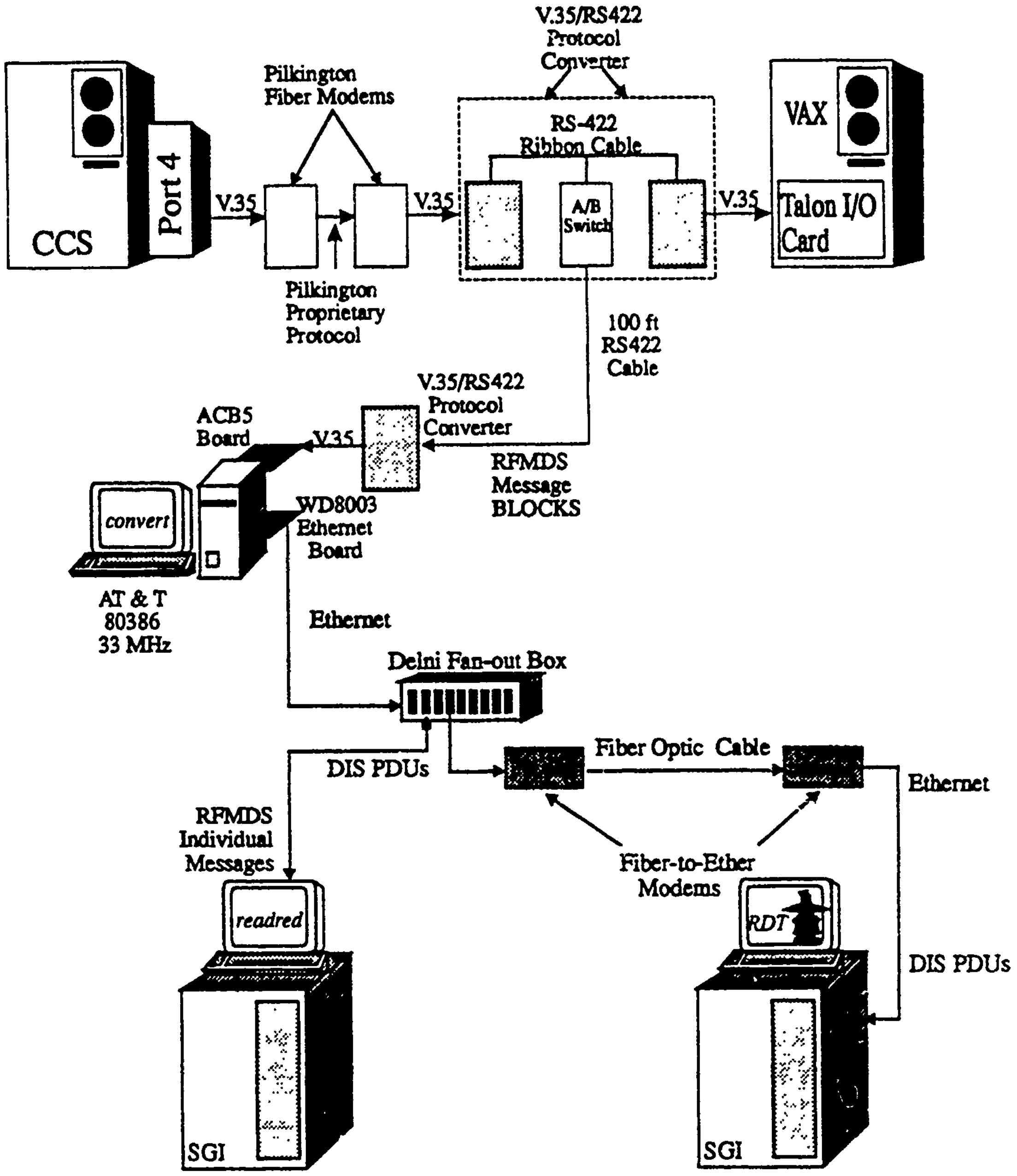

Figure 79. RDT Hardware Configuration

The equipment should be connected as shown in Figure 79. Because the active "T" junction created to tap into the RFMDS data stream directly affects data going to the VAX computer, you must first coordinate with RECOM personnel for approval to install the "T." As long as the V.35/RS-422 converters are powered, there should be no interference with the RECOM operations. If, however, the "T" is deactivated, the 100 ft RS-422 cable serves as a great antennae and brings a lot of noise onto the lines. This noise effectively shuts down the RFMDS-VAX communications link. The "T" must be completely disconnected if it is powered down.

E.1.2 Required Software. All source and executable software for *RDT* is located in the /usr/people/wb directories on the Onyx machines. The /usr/people/wb/bin/RDT directory contains the executable software and the /usr/people/wb/src/RDT directory contains the source code. The following files located in the "bin" directory are needed to run the *RDT*:

- RDTdis (executable)
- afitlogo.rle
- n37w115.small.rgb (texture maps)
- n37w115.small.rgb.attr
- n37w116.small.rgb
- n37w116.small.rgb.attr
- n37w117.small.rgb
- n37w117.small.rgb.attr
- n38w115.small.rgb
- n38w115.small.rgb.attr
- n38w116.small.rgb
- n38w116.small.rgb.attr
- n38w117.small.rgb
- n38w117.small.rgb.attr
- rdtEffectsMgr.dat
- rdtLocalModelMgr.dat
- rdtTerrainMgr.dat
- MissionColors.dat

- rdtSGIDefaultColors.dat
- rdtTypeMap.dat
- readred (executable)
- redflag.cfg
- tub.flt
- xfilter (executable)

In addition to the above files, RFMDS mission data files must be available for use by *Readred* as well as a set of *Multigen* model files for RDT_{dis}. A couple of suitable RFMDS mission data files are located in the /usr/people/wb/bin/RDT/data directory. The mission data files are:

- 15Jul.filter
- set.d3

The current model files located in the /usr/people/wb/bin/RDT/models directory include:

- 707+a.flt
- RedFlagTerrain.round.flt
- a10-a.flt
- b_flame.flt
- dflt_model_1.flt
- f-4_new.flt
- f111.flt
- f15c+a.flt
- f15nolod.flt
- f16+a.flt
- fa18+a.flt
- fireball.flt
- foker_med.flt
- mk82.flt
- newRFTerrainFlat.flt
- sa2_misl+a_2.flt
- sa3_misl+a_2.flt

- sidewinder_plume.flt
- uh60+a.flt
- yf22+a.flt

E.1.3 Setting Up the Data Pipeline. The following subsections explain how to start the software programs necessary to set up the data pipeline from RFMDS to the Computer Communications Component (CCC), Data Translation Component (DTC) and IVC.

E.1.3.1 Convert. Instructions for running the *Convert* program are contained in Appendix B

E.1.3.2 Starting Daemons. In order for *Readred* to send distributed interactive simulation (DIS) protocol data units (PDUs), and for RDT_{dis} to receive PDUs, the AFIT DIS daemons must first be started. Because only one send and one receive daemon may run on a single machine, it is wise to first determine whether the required daemons are active. Do this by using the *ps -elf | grep sgi* command from the UNIX prompt. If the daemons are active, there will be an entry in the resulting display showing the name of the process and any command line options that were used to start the daemon. You should pay particular attention to the '-p' option. This determines which ports are being used by the daemons for the network communication.

If you are setting up the system to run the *Readred* program, you must start a send daemon and optionally start a receive daemon. The receive daemon is only required if you will be receiving individual RFMDS messages from the CCC. If you are setting up the system to run RDT_{dis} you need only start a receive daemon.

To start a receive daemon, change to the /usr/people/wb/bin/Daemons directory and type *sgirecvd -b100 -p3000 &*. This will start a receive daemon using 100 buffers on port 3000. Any port between 2000 and 6000 may be specified. If you desire the receive daemon to receive PDUs from a send daemon on the same machine, use an additional -o switch on the command line when starting the receive daemon.

To start a send daemon, type *sgisendd -b100 -p3000 &*. This begins a send daemon with 100 buffers on port 3000. The same port number assignments apply to the send daemon as described for the receive daemon.

IT IS IMPERATIVE THAT THE SEND AND RECEIVE DAEMONS BE CONFIGURED TO USE THE SAME PORT.

If you determine that one of the daemons is not configured to use the same port as the other, you should stop the daemon and restart it. To stop the daemon type *sgirecvd -q* or *sgisendd -q* depending upon which daemon needs to be terminated.

It is possible to have a daemon terminate and not release the semaphores associated with it. Use a combination of the 'ipcs' and 'ipcrm' UNIX commands to remove the semaphores. See the system documentation for details about these commands.

E.1.3.3 Readred.

Instructions for running the *Readred* program are contained in Appendix C

E.1.4 Configuring RDT.

RDT was designed to allow as many changes to the underlying DIS and RFMDS definitions as possible by using a number of configuration files that are read during the initialization of RDT_{dis}. This same concept is used with *Readred*. The following sections describe the procedures that can be used to make changes to the *RDT* configuration.

E.1.4.1 Model Managers.

RDT_{dis} contains several Model Managers which are responsible for determining the location of any geometric models that are needed for the scene rendering. Four data files are used to communicate to the model managers where the models are located. The format for each of the data files is identical.

When the RDT_{dis} Net Manager determines that a new aircraft model needs to be inserted into the scene, it first uses the Local Model Manager information to find the model instead of using the Net Model Manager information. If the model cannot be located with the Local Model Manager, the Net Model Manager is used to find an appropriate model from the standard models library (/usr/people/wb/models). This mechanism provides an override feature that is useful for replacing the standard library models with ones of your own choosing.

The Effects Model Manager is used to locate the models that are used with munition detonations, while the Terrain Model Manager is used to determine which terrain model will be loaded into the scene geometry list.

Each of the data files used with the model managers is divided into two sections. The first section contains an index number, directory name, and file name for each of the models. The second section contains an object manager type number and the model index numbers from section one that will be associated with the undamaged and damaged entity's appearance, respectively. The object manager's type number is tied directly to an enumerated type definition within the object manager software. This is the mapping that the object manager uses to convert from the multiple-field DIS type definitions to a single number. To change the model that is inserted for a given entity type, simply change the model index numbers in section two. As new DIS entity types are added to the object manager, additional mapping data will need to be inserted into section two. As new models become available, assign new model index numbers and insert the directory/file names into section one of the data files.

The *rdtTerrainMgr.dat* uses one additional field that the other model managers do not use. This is the comment field in section two. If the comment field contains three numbers, these are interpreted as the WGS84 (x, y, z) position of the origin for the terrain model. Terrain models designed to be used with the DIS 'round earth' coordinates must have this origin information included in the comment field. If the origin data is not present, RDT_{dis} interprets the terrain as a flat earth patch and the round-to-flat transformations are not performed. To use other terrain patches with RDT_{dis}, simply comment out the Red Flag terrain entries in the *rdtTerrainMgr.dat* file and add entries for the desired terrain.

E.1.4.2 RFMDS-DIS Player Type Mappings. Two data files are used to map the player type definitions between RFMDS and DIS. The *redflag.cfg* file is used by *Readred* while the *rdtTypeMap.dat* file is used by RDT_{dis}. Both files should remain relatively static and require little modification.

RFMDS Type	DIS Entity Type	Comments
1	(1 2 225 2 1 0 1)	A-4/1
2	(1 2 225 2 1 0 2)	A-4/2
⋮	⋮	⋮
201	(1 2 225 4 1 0 0)	KC-130
202	(1 2 225 4 5 0 0)	KC-135
203	(1 2 225 4 6 0 0)	KC-10
⋮	⋮	⋮
255	(1 2 225 0 0 0 0)	Spare

Table 17. Extracts from redflag.cfg

The *redflag.cfg* file is divided into two principal sections. Section one contains information about the individual RFMDS messages. This data is no longer accessible, but is still parsed by *Readred*. (There are still a number of clean up items that can be performed on *Readred* and this is one of them.) Section two contains the RFMDS to DIS player type mappings. Extracts of items from section two are found in Table 17.

For each of the 255 possible RFMDS aircraft types shown in column one, corresponding DIS entity type definitions and aircraft names are given and shown in columns two and three. RFMDS spare types are mapped to an undefined DIS entity type that causes an object manager error message. The multiple entries for a single RFMDS aircraft in the table correspond to different external store locations used to carry the AIS pod.

The *rdtTypeMap.dat* file is used to convert player types from the DIS definitions to the RFMDS aircraft type definitions. The object manager used within RDT_{dis} defines a single type number for each of the DIS entity types. It is the object manager's number that is passed to RDT_{dis} and used for the conversion. Table 18 illustrates an extract from *rdtTypeMap.dat*.

Column one of the data file contains the first RFMDS type identifier that corresponds to the object manager's type number. Column three contains comments. The object manager's type number defined in common_enums.h and shown in column four is used as the primary search key for the table. If either of the aircraft type identifiers used by RFMDS

RFMDS Start #	RFMDS End #	Comments	ObjMgr Type #
0	0	Spare	0
1	5	A-4	133
6	7	A-6	134
12	19	A-7	135
⋮	⋮	⋮	⋮
201	201	KC-130	145
⋮	⋮	⋮	⋮

Table 18. Extracts from rdtTypeMap.dat

Red	Green	Blue	Forms Color	Comments
0	0	0	0	Index 0 - Black
195	0	0	80	Index 1 - Red
0	0	255	4	Index 2 - Blue
⋮	⋮	⋮	⋮	⋮

Table 19. Extracts from rdtColorTable.dat

or the object manager's type numbers change, both the *redflag.cfg* and *rdtTypeMap.dat* data files must be modified to reflect the changes.

E.1.5 Changing Colors. Aircraft numbers, call signs, and flight path trails are all color coded to correspond to a particular mission role assigned to a player. For example, red is used for the defensive-counter-air player's color, while blue is used for interdiction player. Color assignments for the various mission roles are defined in the Computer Performance Specification Interoperability volume, table 3.4.2-9, sheet 3-100. These assignments can easily be changed by editing two data files–*rdtMissionColors.dat* and *rdtColorTable.dat.*

The *rdtColorTable.dat* file is used to create the RDT_{dis} Color Table. This table contains red, green, and blue (RGB) color values for each of the 11 colors currently defined and used to draw the player numbers, call signs, flight path trails, and plan view symbology. Table 19 represents an extract from the *rdtColorTable.dat* file used to build the Color Table.

Column four of the Color Table contains a color index, used by the *Forms Designer*, that corresponds to the RGB color values in columns 1-3. The RGB values are used with

Mission Role	Color	Comments	
AAR	3	Green	Index 0
ABCC	3	Green	Index 1
ADF	1	Red	Index 2
AMR	4	Yellow	Index 3
⋮	⋮	⋮	

Table 20. Extracts from rdtMissionColors.dat

the SGI graphics language library functions to draw the aircraft numbers, call signs, and flight path trails. Thus, an aircraft with color number '2' will have its flight path trail drawn in blue. The *Forms Designer* color index is used to color the button labels on the 'Mission Code auxiliary control panel to show the mapping between the mission roles and the assigned colors.

Mission roles and the corresponding color assignments are made in the *rdtMissionColors.dat* file. Extracts from that file used to create the Mission Color Table are shown in Table 20.

Mission role values assigned to each aircraft in the RFMDS data range from 0 - 25. For each of the 26 values, a color index is assigned that corresponds to the array index used to reference the colors in the Color Table. As an example, assume aircraft number 10 is assigned an ADF role. The RFMDS data would contain a '2' in the mission identification (ID) field of the maneuver data message. The mission ID would then be translated to the capabilities field in the Entity State PDU by *Readred*. RDT_{dis} would then use the color value '2' as an index into the Mission Color Table. This value would be mapped to index '1' in the Color Table and the RGB triple corresponding to the color red would be used to draw the aircraft's number, '10.'

The number of colors available for mission roles and the current color mappings may be changed by modifying either of the two data files described above. If additional colors are added to the *rdtColorTable.dat* file, you must also determine the *Forms Designer* color index corresponding to the RGB triple you are adding to the file.

E.2 Starting RDT_{dis}

Once the data pipeline is set up, you are ready to run *RDT_{dis}*. Change to the /usr/people/wb/bin/RDT directory and type: *RDTdis*. No additional command line options are required. If the receive daemon is active, you should see a series of initialization messages on the console. If the daemon is not running, *RDT_{dis}* will terminate. Additionally, an eom_arena file in /usr/tmp that was created by another user may cause *RDT_{dis}* to terminate. If the error message displayed on the terminal indicates that this is the case, delete the /usr/tmp/eom_arena file and restart *RDT_{dis}*. To delete the file type: *rm /usr/tmp/eom_arena*. The remaining subsections describe the various controls of the *RDT_{dis}* control panels and the methods used to manipulate views into the Red Flag environment.

E.2.1 Device/Viewport Configuration. The first control panel that appears when *RDT_{dis}* begins execution is shown in Figure 80. *RDT_{dis}*'s default configuration is set such that the Plan View appears on the right graphics viewport and the 3D Views appear on the left. This default configuration may be changed by pressing either the Plan View or 3D View buttons on the configuration control panel.

RDT_{dis} is currently configured to use only an SGI spaceball. Future implementations may elect to add other view modifiers. Until that time, the Future Device buttons are not used.

If changes are made to the default configuration, press the Accept button to incorporate these changes into *RDT_{dis}*. Information about the development of *RDT_{dis}* can be viewed by pressing the About RDT(dis) button. After the desired configuration has been accepted, press the Continue button to move to the *RDT_{dis}* master control panel. The entire screen should go black momentarily while performer initializes the graphics window. Once the master control panel and the terrain appear, *RDT_{dis}* is fully operational.

If no players appear, confirm that *Readred's* send daemon and *RDT_{dis}*'s receive daemon are using the same viewport. Also check to see that *Readred* is still broadcasting.

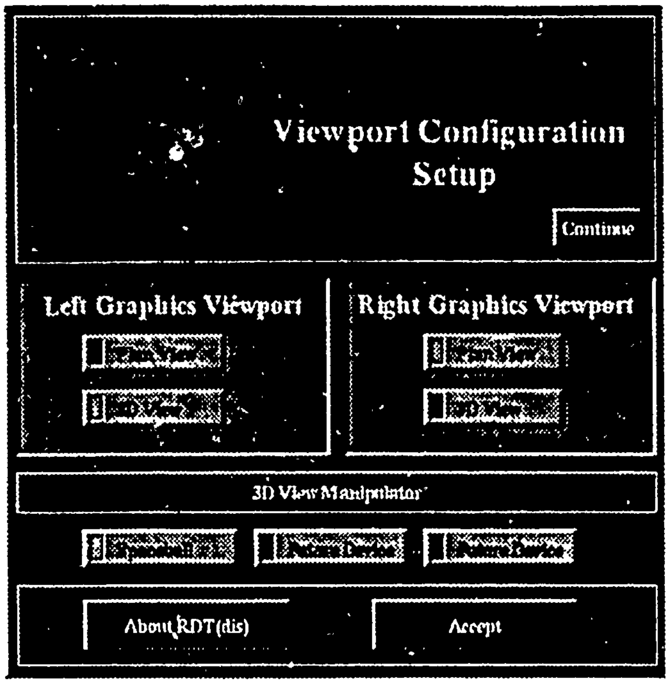

Figure 80. Configure Panel

E.2.2 RDT Controls. The workstation monitor is divided into four quadrants. The 3D View is in the upper left quadrant, and the Plan View is in the upper right quadrant. The lower left quadrant, identified by the AFIT logo, serves as the location for auxiliary control panels and data views. The master control panel, located in the lower right quadrant of the screen, is operated by the user with a standard m[illegible]. The individual controls on any of the control panels are operated by clicking the left mouse button on top of the desired button, slider or positioner. These mouse-based movements are also referred to as selecting/pressing buttons or moving sliders and positioners.

E.2.2.1 Control Panel Sections. Figure 81 shows the master control panel and its logical subdivisions. The main keypad is in the upper left hand corner. The Toggle buttons are in the upper center, and the Expand View and Quit buttons, as well as the clock, are in the upper right hand corner. Directly under the Toggle buttons are the Plan View controls. The positioner crosshairs, in the middle of the control panel, move the Plan View's view position over the terrain in the upper right-hand quadrant. The '+'

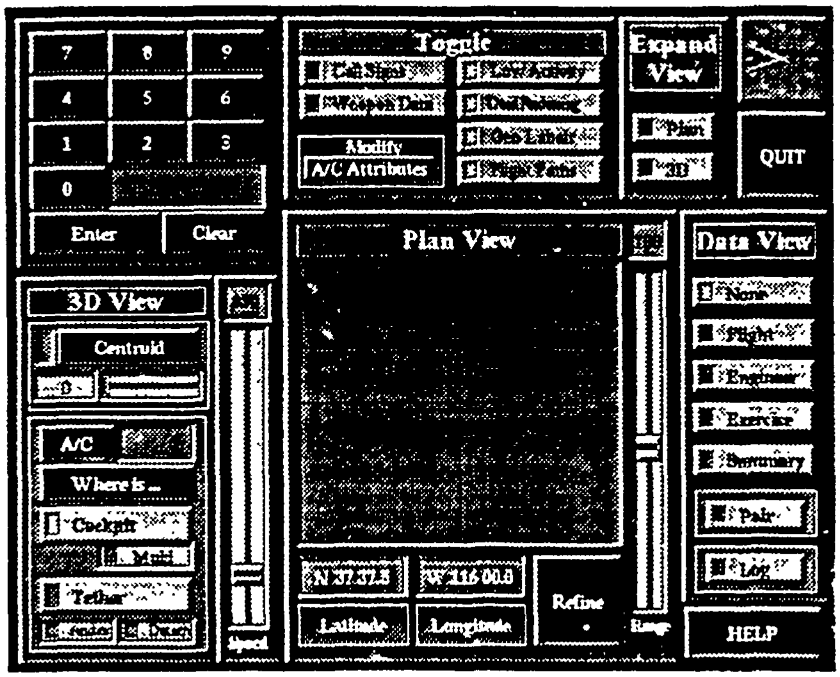

Figure 81. Master Control Panel

symbol indicates the current view position. Along the left hand side of the control panel, you will find the 3D View controls. Located along the right hand side are the Data View select buttons. The Help button is located in the lower right hand corner of the control panel. The subsections that follow describe each of the controls on the master and auxiliary control panels.

E.2.2.2 Help. Pressing the Help button on the master control panel activates the Help auxiliary control panel as depicted in Figure 82. At the present time, only the Function Keys button will display any useful information. The remaining buttons are present only to display a help capability.

E.2.2.3 Keypad. The keypad, located in the upper left hand corner of the master control panel, is used to enter player identification numbers. (Players is a term used to represent either aircraft, weapons, or other network entities.) RDT_{dis} is currently configured to allow up to 500 players. Player number assignments are enumerated in Table 21.

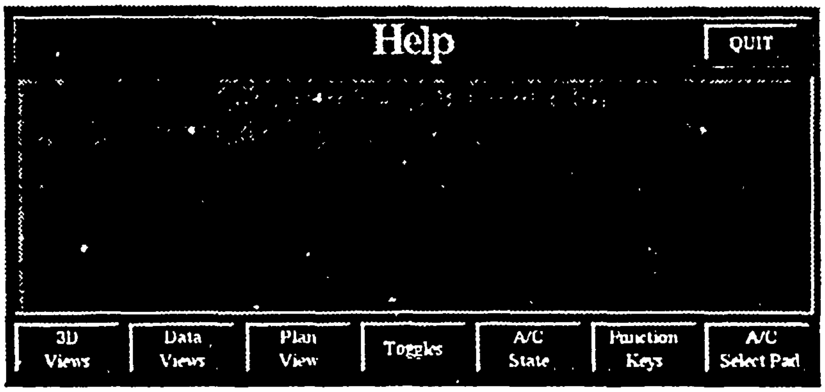

Figure 82. Help Panel

Player #	Player Type
0	Bullseye Player
1 - 255	Red Flag Players
256 - 400	Red Flag Weapons
401 - 500	Network Entities

Table 21. Player Number Assignments

The keypad is activated when any A/C, FTR, or TGT button is pressed. Once activated, the background color of the keypad's information window turns red. To enter a player number, simply click on the numbers with the mouse, and then click on the Enter button. Once the keypad is activated, the Enter button MUST be pressed. As long as the background color of the keypad information window is red, the keypad is still looking for a valid input. Pressing the Enter button changes the background color to green, and deactivates the keypad. If the number entered corresponds to a player that is currently active, the number is accepted. If the player is not currently active, the 'Inactive A/C' message appears to inform you that the number you entered was ignored. (Note: As long as the keypad background is red, many of the other controls on the panel are locked out. Press the keypad Enter button to unlock.)

E.2.2.4 Spaceball. The SGI spaceball, which communicates with the workstation on port 2, can be used to modify the 3D Views in RDT_{dis}. The spaceball offers six degrees of freedom: translation forward and backward, left and right, up and down, as well as pitch up and down, roll left and right, and twist left and right. In general, pushing forward on the spaceball moves the view position forward along the view player's orientation vector, and pulling backward on the spaceball moves the view in the opposite direction. Pulling up and pushing down on the spaceball correspond to vertical movements of the view position. Twisting the spaceball left or right rotates the view around the view position's local vertical coordinate axis. Roll inputs to the left and right are currently ignored in this application, however, rolling the spaceball forward causes the view to pitch down and rolling the spaceball backward causes the view to pitch up. The view position translations and rotations just described are only applicable if the view orientation is 'straight and level.' Once the view orientation changes, such as when the 3D View is attached to an aircraft performing some 3-dimensional rolling maneuver, the conventional notions of movement with the spaceball change. The next paragraph explains why.

The 3D View's position and orientation is determined from the currently selected 3D View. This position and orientation in the world coordinate reference frame becomes a local reference frame for inputs taken from the spaceball view modifier. Thus, a forward movement on the spaceball translates the view forward along the view orientation vector. For example, assume we are attached to an aircraft pointing straight down in the world coordinate reference frame. Pushing forward on the spaceball, moves the view forward in the local reference frame, but that forward movement translates to a downward movement in the world coordinate reference frame. Pulling the spaceball backward translates the view in the local reference frame backward but that translates to a vertically upward movement in the world coordinate reference frame. If this all seems a little confusing, here is another example to clarify how the spaceball modifies view positions and orientations.

Presume that the tether view has been selected and that you would like to view the front of the aircraft. Pushing forward on the spaceball will translate the view position forward along the aircraft's heading until the desired position in front of the aircraft is reached. Twisting the spaceball left or right will rotate the view around until the front of the aircraft is visible. So far, everything seems normal. If, however, you wanted to move the view position closer to the front of the aircraft, you would have to pull the spaceball backward, instead of pushing the spaceball forward, to get the desired movement.

Twisting inputs on the spaceball do not modify the base orientation of the 3D View. Translation inputs from the spaceball are combined with the base orientation of the 3D View to determine the modified view's final orientation.

Because the view modifications are made in the local reference frame, the spaceball has limited usefulness when the 3D View position is not oriented 'straight and level.' A view reset button located on the forward surface of the spaceball nullifies any previous spaceball inputs. The spaceball is best suited for the Anchor and Detach Views and has limited utility with the other views.

Buttons 1-4 on the front of the spaceball support are used to increase/decrease the translation and rotation sensitivity of the spaceball. Pressing button 1 or 3 increases the spaceball sensitivity for translations and rotations respectively. Pressing button 2 or 4 decreases the sensitivity.

E.2.2.5 Plan View. Two controls associated with the Plan View are the the Positioner window and the Range slider, located immediately to the right of the Positioner. The number at the top of the Range slider represents the number of miles "visible" in the Plan View. The Range slider is used to move the view position vertically up and down and produces the same effect as the zoom feature on a camera. Moving the Range slider to the top gives you the widest possible view, with a maximum horizontal range of 200 miles.

The intersection of the crosshairs in the Positioner window represent the relative position of the cursor, '+', in the Plan View. Clicking the mouse anywhere in the Positioner window moves the crosshairs to the mouse position, with a corresponding movement of the cursor in the Plan View. Latitude and Longitude windows, located below the Positioner window, display the world coordinates of the cursor in the Plan View.

Pressing the Refine button, to the right of the Latitude and Longitude windows on the master control panel, brings up the Refine auxiliary control panel illustrated in Figure 83 and deactivates the Plan View controls on the master control panel.

The purpose of this auxiliary control panel is to allow finer movements of the cursor over the terrain. The controls associated with this control panel are identical to the Plan

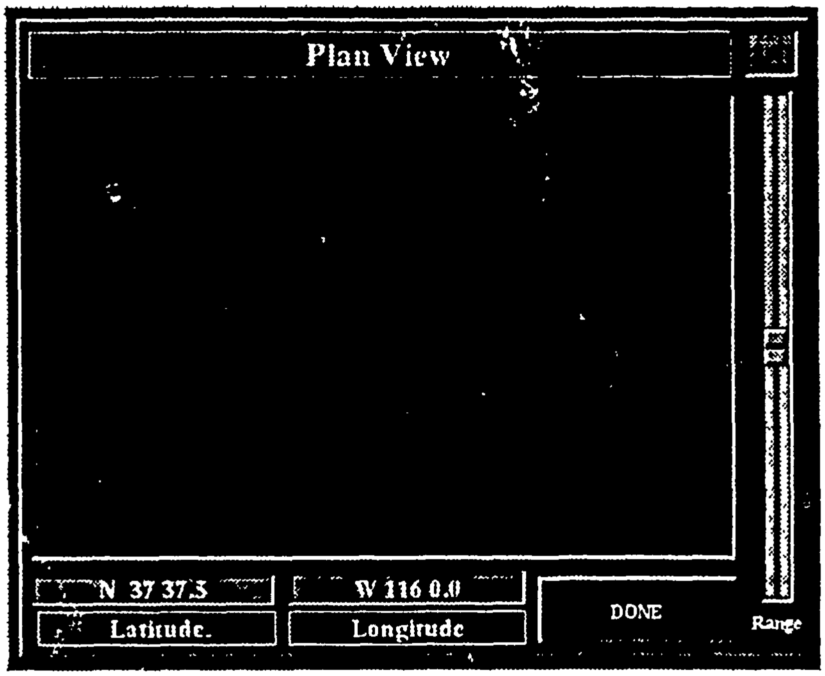

Figure 83. Plan View Panel

View controls on the master control panel. To remove the auxiliary panel, simply press the Done button on the lower right side.

E.2.2.6 3D Views. There are three main 3D Views–Centroid, Cockpit, and Tether. The Tether View has two subviews: Anchor and Detach. Use the buttons on the left side of the master control panel to switch between these views. A green light next to the button indicates which view is currently active. Only one may be active at a time.

Centroid View. The Centroid View focuses the viewing volume upon the selected players and automatically repositions the view volume to keep these centroid players in sight at all times. Pressing the Centroid button brings up the Centroid auxiliary control panel, activates the heading and elevation sliders (located just below and to the right of the Centroid button, respectively), and deactivates the Cockpit and Tether buttons.

When the Centroid auxiliary control panel of Figure 84 is visible, you may select up to four players upon which to center the view. These players may be selected in any

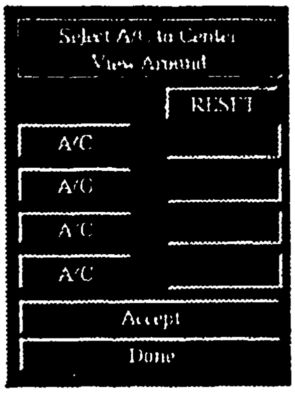

Figure 84. Centroid View Control Panel

order. First, press an A/C button and use the keypad to enter the player number. Repeat this procedure for all four players. Pressing the Reset button erases player numbers from the Centroid auxiliary control panel windows. Once a player number has been selected and validated by the keypad, the background color of the window containing the number changes from green to lavender. When all selections have been completed, press the Accept button to change the view in the upper left quadrant of the screen. The view will not change until the Accept button has been pressed. After the Accept button has been pressed, the background color of the player numbers changes from lavender back to green to let you know that the view has been accepted.

The Centroid View's position and orientation can be changed with the heading and elevation sliders as long as the Centroid auxiliary control panel is visible. The horizontal heading slider can be used to modify the view orientation. The default view heading is 000 or North. A window immediately to the left of the heading slider indicates the current view direction. Moving the slider to the right moves the heading from 000 to 359. A value of 090 in the window indicates that the view is oriented to look to the East.

The Elev slider, which doubles as the Speed slider for the Detach subview, can be used to change the Centroid View's position from directly overhead to a position level with the center coordinate of the selected centroid players. A window at the top of the Elev slider indicates the current view elevation. The default elevation is 090, or directly overhead.

After you have finished selecting the centroid players and have set the Centroid View's heading and elevation, press the Done button to hide the Centroid auxiliary control panel, deactivate the heading and elevation sliders, and reactivate the Cockpit and Tether buttons.

Cockpit View. The Cockpit View is attached to a position a fixed distance from the origin of the player's geometric model. This view allows you to see the exercise from a pilot's point of view. The generic head-up display (HUD) is visible in this view.

To select the Cockpit View, press the Cockpit button on the left side of the master control panel. Press the A/C button, located immediately above the Cockpit button, to attach to a player.

Tether View. The main Tether View is attached to a position directly behind a player along the flight path vector. To select the Tether View, press the Tether button on the left side of the master control panel. Press the A/C button to tether to a player.

In the Anchor subview, the view position is determined by the crosshairs on the Plan View Positioner window. The spaceball allows you to modify your view orientation. The Anchor subview is useful for setting a view position on or near the ground at a target location. This allows analysis of a flight's geometry, spacing, and timing as it attacks a target. To select the Anchor subview, press the Anchor button, located below the Tether button, while the Tether View is activated.

Pressing the Detach button, located immediately below the Tether button, allows you to "fly" anywhere through the environment. Use the spaceball to control your heading and pitch and the Speed slider on the right side of the 3D View control area for your airspeed. The airspeed is indicated in the window directly above the Speed slider and may range from -90 knots to 3000 knots. An asterisk on the Plan View screen shows your location.

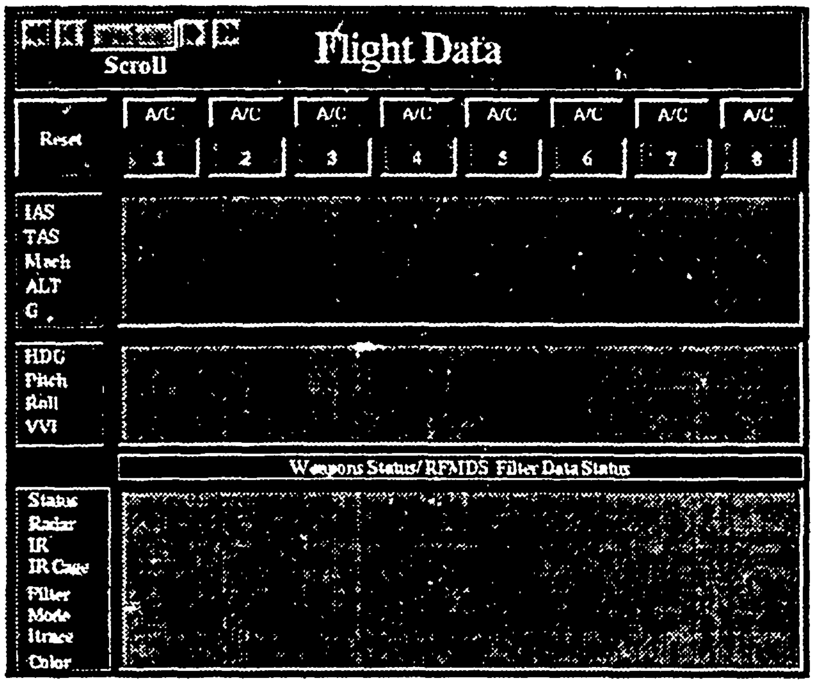

Figure 85. Flight Data View Panel

E.2.2.7 Data Views. Press one of the four Data View buttons on the right side of the master control panel to bring up specific data about an individual player and its identity. The selected Data View auxiliary control panel will be displayed in the lower left hand quadrant of the screen. The None button hides any of the Data views.

Flight and Engineering Data Views. The Flight and Engineering Data View auxiliary control panels are identical in the way they are operated. The only difference between the two Data Views is the information contained in the bottom portion of the display window. See Figure 85.

In order to review flight or engineering information about a player, press one of the A/C buttons at the top of the panel. The default setting displays information about player numbers 1-8. Pressing a single arrow button to the right or left of the scroll window at the top of the panel increases/decreases the number in the scroll window by one. This number corresponds to the player number in the leftmost column of the display. Selecting the double arrow button increases/decreases this number by 8.

The information in these Data Views is primarily intended for use with the Red Flag high- and low-activity aircraft. Therefore, when scrolling, you will notice that the allowable values are 1-40 and 100-200. Information about players outside of these ranges require you to manually enter the player number by pressing one of the A/C buttons and using the keypad.

Exercise Data View. The Exercise Data View provides the following information about a player: identity, call sign, time of last position update, player type, and mission role. This view only provides a "snapshot" of the information available at the time the Exercise button is pressed and is not continuously updated. Press the Refresh button at the top left of the Exercise Data View auxiliary control panel to update the information. Data regarding any player for which PDUs have been received will be shown. In order to remove information about inactive/deleted players, press the Remove Inactive button on the upper right of the panel. The scroll bar, located to the left of the display area, is visible only when the amount of information exceeds the panel's display capacity. Move the scroll bar up and down to reposition the information displayed. An illustration of the Exercise Data View Panel is shown in Figure 86.

Weapons Summary Data View. The Weapons Summary Data View auxiliary control panel shows a synopsis of all of the fire and detonation events occurring during an exercise session. Unlike the Exercise Data View, the Weapons Summary View is updated each time a new event is received. Information about these events includes: time, type of event, weapon launcher and target, hit or miss and a reason for miss code, bearing, range, altitude difference, and closing velocity at the time of the event. This data can be saved to a disk file by pressing the Log button in the lower right corner of the master control panel. The Erase button in the upper right corner of the panel removes all weapons events from the display. See Figure 87 for a depiction of this control panel.

RFMDS identifies 90 reasons for a miss in one of its weapons simulations. Tables 22, 23, 24, and 25 contain an itemized list of these reasons and their respective codes.

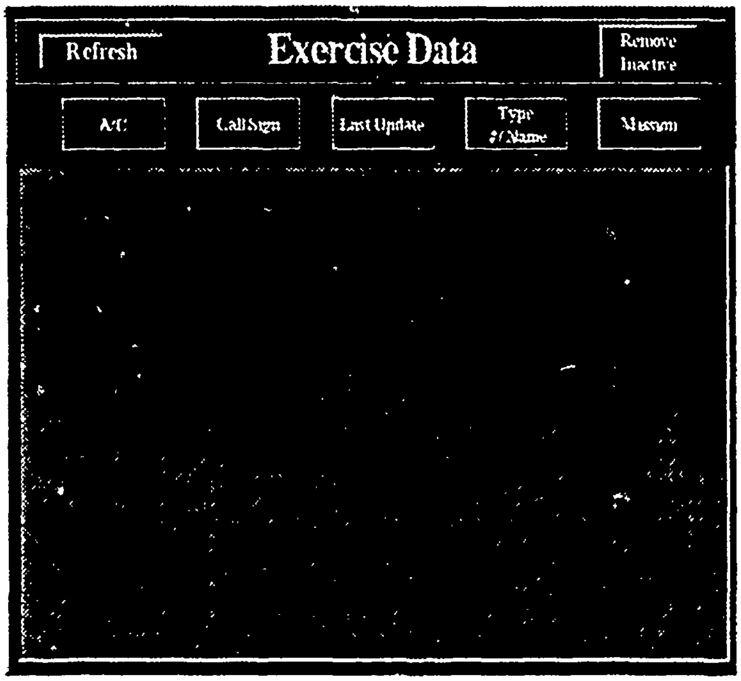

Figure 86. Exercise Data View Panel

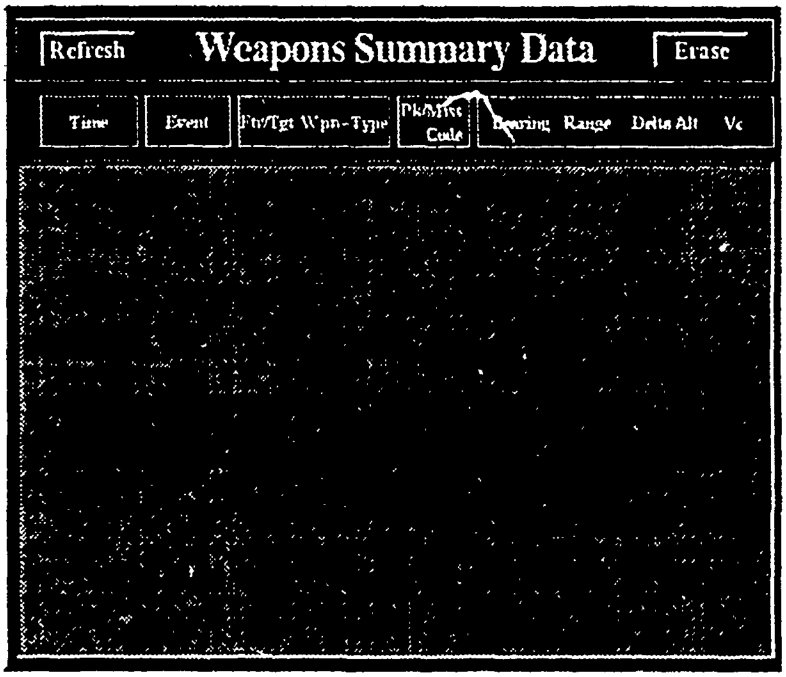

Figure 87. Weapons Summary Data View Panel

Miss Value	Miss Description
0	Hit
1	Excessive lag heading
2	Excessive lead heading
3	Inside minimum range
4	Outside maximum range
5	Insufficient missile velocity
6	Gimbal limit exceeded
7	Seeker lost lock, excessive LOS rate
8	Exceeded missile maneuver capability
9	Interceptor radar not locked at launch
10	Insufficient closing velocity at time of launch
11	SEAM tone absent (training Sidewinder)
12	Sensor masked by terrain
13	Missile acquired the sun
14	Launcher radar broke lock (radar lock information available). Launcher radar outside gimbal units (radar lock information unavailable)
15	Replica sensor defeated (vice simulated sensor)
16	Intercept time less than safe arming
17	Inadequate missile settling time
18	Exceeded allowable miss distance (missile simulation). Very large aiming error (gun simulation)
19	Outside of gun or platform operating range (not now implemented)
20	Exceeded maximum flight time
21	Aiming lagged target (guns only)
22	Spare
23	SEAM angle or tone not available/incorrect
24	IR signal strength adversely affected seeker performance at launch
25	No remaining ammunition (guns only)
26	Possible clutter
27	Rate of change of closing velocity too large for missile to track doppler
28	Wide doppler gate selected (radar missiles only)
29	Insufficient closing rate
30	Dogfight mode improperly set

Table 22. Weapon Simulations Miss Reasons (3:A-131–A-137)

Miss Value	Miss Description
31	Probabiiistic miss
32	No IR target capability in front hemisphere
33	IR signal strength adversely affect seeker performance during flight
34	High launcher angle of attack/sideslip adversely affected missile performance
35	Incorrect or out-of-range input
36	Outside missile seeker range
37	Insufficient lead heading
38	Seeker field-of-view exceeded
39	Excessive angle off tail
40	Incorrect switch settings
41	Exceeded launcher G limit
42	Control burnt out
43	Radar status (threat radar turned off during missile flight)
44	Minimum angle with horizon attained
45	Missile intercepted the ground
46	Gun pointing error
47	Missile model invalid
48	Sensor elevation gimbal lock
49	Roll gyro gimbal lock
50	Missile locked onto flare
51	Missile decoyed by chaff
52	Missile decoyed by expendable jammer
53	Missile decoyed by onboard jammer
54-60	Spare

Tabie 23. Weapon Simulations Miss Reasons, Continued (3:A-131–A-137)

Miss Value	Miss Description
61	Sensor killed before weapon intercept
62	No target
63	Simulation not available
64	Missile failed to leave the rail; determined by comparison the probability of a launch failure for a particular missile and a random number
65	Out of AIM-7 missiles
66	Out of AIM-9 missiles
67	Target aircraft has already been scored dead when fighter shoots
68	Target is scored dead after fighter shoots, but before missile intercept
69	AIM-9 shot is attempted when system detects no coolant has been present
70	AIM-7 missile is terminated when target ceases to be illuminated because fighter is subsequently scored dead or changes illumination mode (CW-PD)
71	Sufficient tone present, but no target aircraft detected
72	Insufficient I.R. tone present
73	Out of anti-radiation missiles
74	No weapon in inventory slot
75	Threat out of SAM's
76	Launch aborted due to bad aircraft track
77	AIM-7 trigger received but aircraft not carrying AIM-7
78	AIM-9 trigger received but aircraft not carrying AIM-9
79	ARM trigger received but aircraft not carrying ARM
80	Phoenix trigger received but aircraft not carrying Phoenix

Table 24. Weapon Simulations Miss Reasons, Continued (3:A-131–A-137)

Miss Value	Miss Description
81	No Phoenix stores remaining
82	AIM-9 arm switch off
83	Out of weapon
84	No displayable event
85	Simulation terminated successfully
86	Out of laser guided bombs
87	Laser guided bomb trigger received but aircraft not carrying laser guided bombs
88	No laser designated target or laser guided bomb outside of laser cone
89	Invalid Phoenix missile mode at launch
90	AIM-54 missile is terminated when target ceases to be illuminated because fighter is subsequently scored dead or changes illumination mode

Table 25. Weapon Simulations Miss Reasons, Continued (3:A-131–A-137)

E.2.2.8 Pair Data. The Pair Data View auxiliary control panel is selected by pressing the Pair button at the bottom of the Data View control area on the master control panel. Activating this panel provides the following information about two players: range, bearing, closing velocity, altitude difference, aspect angle, and antenna train angle. Press the FTR and TGT buttons at the top of the display to pair two players and initiate the display calculations. The Reset button is used to erase all players from the FTR/TGT windows. Press the Pair button on the master control panel a second time to remove this panel. An illustration of this panel is contained in Figure 88.

E.2.2.9 Event Logging. Weapons events may be recorded to a disk by pressing the Log button, located in the lower right hand corner of the master control panel. A File Select panel will appear in the center of the screen. Press the Enter button on the computer keyboard to accept the default file name shown in the File Select panel or use the mouse and keyboard to enter a new file name. A green light will appear next to the Log button to indicate that this function is active. The panel disappears when the Enter button on the keyboard or the Cancel button on the File Select panel is pressed. To disable event logging, press the Log button a second time.

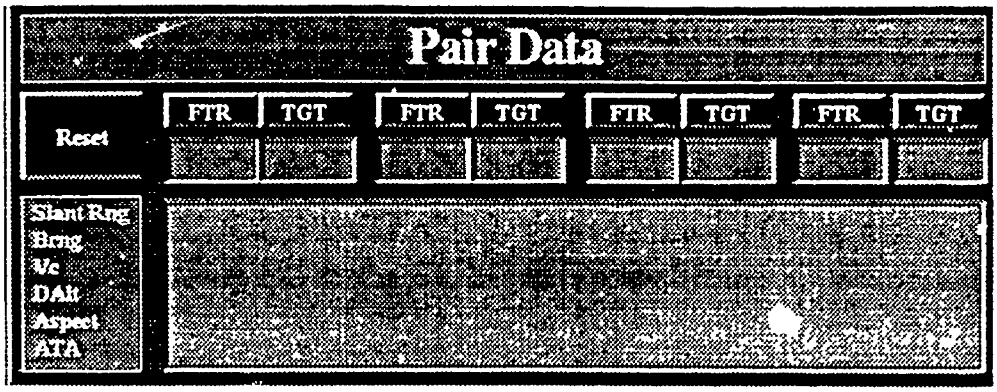

Figure 88. Pair Data View Panel

E.2.2.10 Aircraft Attributes. The Aircraft Attribute auxiliary control panels allow you to change the length of the flight path history trails and the color assignment of a particular player. To activate this panel, press the A/C Attributes button located above the Plan View controls.

To change the length of the flight path history trails, move the slider to the right or left and press the Accept button. The length of the trails, measured in seconds, may range from 0-60.

To change a player's mission color, press the A/C button and enter the player number. Once this number has been verified by the keypad, the Mission Code auxiliary control panel will be displayed. Press the button for the desired mission role. Pressing the Done button removes both panels. Figure 89 shows these panels.

E.2.2.11 View Toggles. Several of the display features and functions associated with RDT_{dis} can be turned on and off with the Toggle buttons located at the top of the master control panel.

Press the Call Signs button to switch the player identification between the player number and an alpha-numeric call sign.

Press the Weapons Data button to display the four most recent weapons events in the lower right corner of the 3D or Expanded View.

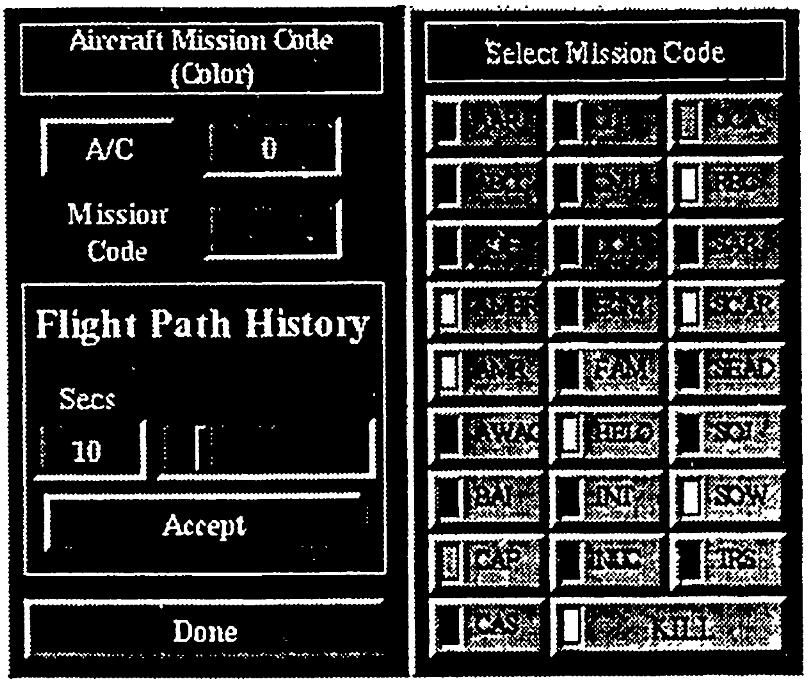

Figure 89. Attribute Panels

Pressing the Low Activity button effectively hides tLe low-activity players from both the Plan and 3D Views by removing the player identification number/call sign and down-scaling the model size. (Any player number other than 1-36 is considered a low-activity player for this toggle.) A second press of the button restores the display of low-activity players.

Press the Dead Reckoning button to turn dead reckoning off and on. This does not affect *Readred's* dead reckoning computations.

Press the Geo Labels button to turn the geographic reference point labels on and off. These include range boundaries, restricted areas, and single-letter ground reference points.

Pressing the Flight Paths button turns the flight path history trails on and off. This does not affect flight path trails used to locate players selected for focus in the centroid view.

E.2.2.12 Time Reset. The range time displayed in the lower left hand corner of both the Plan and 3D Views is derived from information embedded in the Entity

State PDUs. This time represents the most recent time received and is not a real-time clock. The DIS standard requires that active entities broadcast Entity State PDUs at least once every five seconds; therefore, the displayed range time may jump in increments of up to five seconds. A hidden button on the clock face in the upper right hand corner of the master control panel will allow you to reset the range time to 00:00:00:00. Press the center of the clock face to activate the button.

E.2.2.13 Expanding Views. To expand views from the 1/4 screen mode to 1/2 screen mode, press one of the expand buttons in the upper right hand corner of the master control panel, next to the clock. These are toggle buttons. Press again to return to 1/4 screen mode. If you have pressed the expand button and are in the 1/2 screen mode, you may also expand one level further to full screen mode by pressing the F3 function key on the computer keyboard. This action places the master control panel underneath the full screen graphics window. All other control panels remain visible. Pressing the F4 function key returns you to the 1/2 screen mode and places the master control panel on top of the graphics window and any other auxiliary panels. In most cases, reselecting the appropriate auxiliary panel button will make it visible again.

When you are in the full screen mode, pressing the F1 function key on the computer keyboard will bring the master control panel into view without resizing the graphics window. Pressing the F2 function key will hide the master control panel. This is useful for making momentary control inputs while allowing you to remain in the full screen mode.

E.2.2.14 Quit. You can quit RDT by either using the Quit button on the upper right hand side of the master control panel under the clock, or by pressing the Escape button on the computer keyboard.

E.3 Performing Analysis Tasks

Directions for analyzing various types of data and performing analysis tasks not previously described are included in this section.

E.3.1 Bullseye Analysis. A bullseye point is a geographic reference point known to the participants in an exercise. Airborne and ground controllers use this known reference to point out enemy or unknown aircraft to friendly players. As an example, the call, "Bandits, bullseye, 210, 20" indicates that enemy aircraft are located at a range of 20 nautical miles from the bullseye point on a magnetic bearing of 210 degrees. All friendly aircraft may then calculate the position of the bandits relative to their position. The bullseye point is displayed only in the Centroid View with player number '0' selected.

The position of the bullseye player is set by using the plan view positioner and referencing the latitude/longitude displays. Alternatively, the bullseye position may be set by referencing the terrain texture map and setting the position over a known ground reference.

Pair data is also available for the bullseye player. Just enter '0' as the FTR on the pair data form. Entering an active player into the TGT position on the pair data form will then enable the pair calculations and provide the needed bearing and range from the bullseye point to the TGT.

E.3.2 Data Reliability. The reliability of the RFMDS tracking systems can be determined by referencing the Flight Data View auxiliary control panel and noting three fields: filter, mode, and Itrace. The *filter* field indicates whether the position tracks processed by the RFMDS data filters are 'OK' or 'Unreliable.' The *mode* field gives an indication of the accuracy of the aircraft's position by showing whether the position is being calculated normally or is being predicted. The *Itrace* field is another method used to determine the accuracy of the data. Reference Table 26 for a description of the individual Itrace values.

E.3.3 Vertical Airspace Depiction. A vertical depiction of the airspace is useful to determine the altitude separation between aircraft and to ensure that altitude block assignments are not violated. Either the Centroid View or the Tether View may be used to show a vertical depiction.

Values	Conditions of Filter Internal Operation
0	Filter output data all available
1	Filter is in multilateration mode
2	Ranging matrix not positive definite
3	Tracking aided by radar altimeter
4	Tracking on less than three ranges
5	Tracking on radar measurement and less than two ranges
6	Tracking on barometric measurement and less than two ranges
7	Downlink data consistency failure
8	Filter output data consistency failure
9	All ranges and baro and radar altitude fail filter editing
10	Total filter blanking (downlink and ranging failure)
11	Data unreliable after initialization
12	Uplink data failure
13	All ranges fail gross checks and baro and radar altitude invalid
14	Baro-aiding required but not available
15	TIS input buffer checksum error
16	Excessive uplink parity failure
17	Excessive roll rate during initialization or uplink correction limited to what initialized pod will accept
18	Tracking aided by barometric measurement
19	No downlink air data
20	Aircraft off range with ITRACE of zero
21	Possible mirror image solution removed from DDS until confirmed
22	Terrain map z used as a measurement to update filter z
98	Whole value uplink requested by AIS
100	Filter restart, critical cycle
101	Filter restart, excessive blanking
102	Filter restart, excessive uplink data failures
103	Filter restart, excessive uplink parity failures
104	Filter restart, excessive downlink data failures
105	Filter restart, excessive output data consistency failures
106	Filter restart, excessive roll during initialization
108	Filter restart, excessive tracking on altitude measurement only
200	New aircraft, begin initializa ion

Table 26. RFMDS Itrace Values (3:A-72)

If the Centroid View is used, the view elevation slider can be set to the zero degree position. This positions the centroid viewpoint level with the exact vertical midpoint of the selected aircraft. Thus, some of the aircraft will appear above the viewpoint position and some below. This may be sufficient for a small set of the aircraft immediately visible in the centroid view volume.

If the Tether View is used, the Anchor subview must be selected. The anchor viewpoint elevation is initially fixed at approximately 7400 ft, but may be modified by the spaceball. To use the Anchor subview for a vertical position first set the location of the anchor viewpoint by mcving the Plan View positioner and then twist the spaceball to rotate the view volume until the desired players are within the field of view. If the bullseye has previously been selected as one of the centroid players, the bullseye position will need to be reset if the anchor viewpoint position was modified.

E.3.4 Locating Aircraft. The simplest means of locating an aircraft is to use the Where is... button. First, enter the number of the aircraft to be located into the keypad by pressing the A/C that is immediately above the Where is... button. Second, press the Where is... button. The Plan View's position is moved to a point directly over the desired player's prsition.

E.3.5 Trouble Shooting. As with many programs, anomalies may occur. This subsection contains possible explanations for some of the anomalies you may notice during a session with *RDT*. Table 27 may be used as a trouble shooting guide. It contains a list of all of the known irregularities encountered to date.

Some of the problems listed in Table 27 are a direct result of the UDP/IP communications protocol at the heart of the DIS standard. Some PDUs are not recovered from the network and thus, notification of fire or detonation events may not be received. No definitive solution has yet been found to overcome this problem.

Problem	Explanation	Corrective Action
Weapon update time shows 00:00:00:00	Fire PDU not received Non-Red F!ag player	None None
Weapon never detonates	Detonation PDU not received	None
Weapon originates at terrain origin	Launching platform unknown	None
Aircraft orientation is incorrect	Most likely cause is bad RFMDS data. See Table 26 for explanations of Itrace values	Confirm RFMDS data reliability with the Flight Data View
Flight path trail appears ahead of the aircraft	Player is decelerating and the dead reckoning calculations are predicting a new position ahead of the aircraft	None
Aircraft not visible but but data is available	Flat-earth terrain patch is used with round-earth data	Restart with a round-earth terrain patch
Program terminates abnormally with a Performer fatal error: uspsema(34)	Performer v1.2 anomaly	Consult SGI
Black polygons become visible in scene	Performer v1.2 anomaly	Consult SGI
Earth/Sky just above horizon blanks out	Performer anomaly	Consult SGI This problem is related to the position of the far clipping plane

Table 27. Trouble Shooting Guide

Bibliography

1. Alluisi, Earl. "The Promise of Interactive Networking: New Levels of Training Readiness in Peacetime." *Proceedings Interactive Networked Simulation for Training.* 3. 1989. Summary of Keynote Address.
2. Ault, Frank W. "The Ault Report Revisited," *The Hook*, 36–39 (Spring 1989).
3. Cubic Defense Systems. *Appendix A Message Catalog of the Computer Program Performance Specification for the Control and Computation Subsystem (CCS)*, March 1992. SP1507-198C Ch 1.
4. Erichsen, Matthew. *Weapon System Sensor Integration for a DIS v2.0.3 Compatible AFIT Virtual Cockpit.* MS thesis, Air Force Institute of Technology, 1993.
5. Foley, James D., et al. *Computer Graphics: Principles and Practice* (2 Edition). Addison-Wesley Publishing Company, 1990.
6. Gerhard, William. *Virtual Cockpit - Weapon System Integration for the AFIT Virtual Cockpit.* MS thesis, Air Force Institute of Technology, 1993.
7. Harvey, Edward P. and Richard L. Schaffer. "The Capability of the Distributed Interactive Simulation Networking Standard to Support High Fidelity Aircraft Simulation." *Proceedings of the 13th Interservice/Industry Training Systems Conference.* 263–271. 1991.
8. Industrial Computer Source, San Diego, CA. *Industrial Computer Source Product Manual*, 1992. Model ACB5 Reference Manual No. 00750-103-10A.
9. Kunz, Andrea. *A Virtual Environment for Satellite Modeling and Orbital Analysis in a Distributed Interactive Simulation.* MS thesis, Air Force Institute of Technology, 1993.
10. Locke, John. "An Introduction to the Internet Networking Environment and SIMNET/DIS." unpublished, Naval Postgraduate School, February 1993.
11. McLendon, Patricia. *IRIS Performer Programming Guide.* Mountain View, CA: Silicon Graphics, I c., 1992.
12. Miller, Harold G. "Wargaming Networks for Training." *Proceedings Interactive Networked Simulation for Training.* 44. 1989.
13. Overmars, Mark H. *Forms Library v2.1 - A Graphical User Interface Toolkit for the Silicon Graphics Workstations.* Utrecht University, Netherlands, Department of Computer Science, 1992. email markov@cs.ruu.nl.
14. Pope, Arthur R. and revised by Richard L. Schaffer. *The SIMNET Network and Protocols.* Technical Report Report No. 7627, BBN Systems and Technologies, June 1991. Prepared for Advanced Research Projects Agency (ARPA).
15. *ReadRed v1.0.* Air Force Institue of Technology, Wright-Patterson AFB, OH, 1993. Data Translation Component, Computer Software.
16. Riggs, Roger D. "Red Flag–Realistic Training in a Simulated Combat Environment." *Interactive Networked Simulation for Training.* 33–38. 1989.

17. Rumbaugh, James, et al. *Object-Oriented Modelling and Design.* Englewood Cliffs, NJ: Prentice Hall, 1991.

18. Sheasby, Steven M. *Management of SIMNET and DIS Entities in Synthetic Environments.* MS thesis, Air Force Institute of Technology, 1992.

19. "Distributed Simulation." Brochure published and distributed at ACM SIGGRAPH convention, Anaheim, CA, Aug 1993.

20. Snyder, Mark. *ObjectSim - A Reuseable Object Oriented DIS Visual Simulation.* MS thesis, Air Force Institute of Technology, 1993.

21. Soltz, Brian. *Graphical Tools for Situational Awareness Assistance for Large Synthetic Battle Fields.* MS thesis, Air Force Institute of Technology, 1993.

22. SRI International (under the authority and direction of the Aeronautical Systems Division, Deputy for Range Systems), Eglin Air Force Base, Florida. *Red Flag Measurement and Debriefing System (RFMDS): User's Handbook*, May 1992.

23. *Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications.* Technical Report IST-CR-93-15, Institute for Simulation and Training, May 1993. Version 2.0, Third Draft.

24. Stevens, W. Richard. *UNIX Network Programming.* Englewood Cliffs, New Jersey: Prentice Hall, 1990.

25. Tanenbaum, Andrew S. *Computer Networks* (2 Edition). Englewood Cliffs, NJ: Prentice Hall, 1989.

26. Tanenbaum, Andrew S. *Structured Computer Organization* (3 Edition). Englewood Cliffs, NJ: Prentice Hall, 1990.

27. Thorpe, Jack A. "Warfighting with SIMNET–A Report from the Front." *Proceedings of the 10th Interservice/Industry Training Systems Conference.* 127–135. 1988.

28. University of Central Florida Institute for Simulation and Training. *Distributed Interactive Simulation: Operational Concept*, January 1992. DRAFT.

29. Wilson, Kirk. *Synthetic Battle Bridge: Information Visualization and User Interface Design Applications in a Large Virtual Reality Environment.* MS thesis, Air Force Institute of Technology, 1993.

Vita

Major Gardner was born in Cedar City, Utah on 26 June 1956. Following his first year of college at Brigham Young University (BYU), he completed a two-year mission for the Church of Jesus Christ of Latter-Day Saints to southern Germany. Afterwards he returned to BYU and graduated in 1980 with a Bachelor's degree in Computer Science as well as a commission in the United States Air Force as a second lieutenant. While waiting to report to active duty, he worked as a computer programmer for Eyring Research Institute, Inc in Provo, Utah. As a distinguished graduate from Undergraduate Pilot Training at Reese AFB, Texas in 1982, Major Gardner was selected to return as a T-38 Instructor Pilot. In 1987 Major Gardner completed transition training in the F-4E/G at George AFB, California, and was then assigned to Spangdahlem AB, Germany as a Wild Weasel Pilot/Instructor Pilot. As Flight Commander in the 81 FS, he deployed to Bahrain and flew 36 combat missions over Iraq and Kuwait in support of Operation Desert Storm. Among other decorations, he is a recipient of the Distinguished Flying Cross. In 1992 Major Gardner was selected to attend the Air Force Institute of Technology to complete a Master of Science degree in Computer Systems. He is married to Patricia Ann (Klein) Gardner of Grand Prairie, Texas. They have three children: Carolyn, Robert and Heidi.

Permanent address: 121 West 3500 South
Bountiful, Utah 84010

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis

4 TITLE AND SUBTITLE A DISTRIBUTED INTERACTIVE SIMULATION BASED REMOTE DE-BRIEFING TOOL FOR RED FLAG MISSIONS	5. FUNDING NUMBERS
6 AUTHOR(S) Michael T. Gardner	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/93D-09
9 SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES) ARPA/ASTO 4301 N. Fairfax Ave, Suite 200 Arlington, VA 22203	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution Unlimited	12b DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Air Force leaders, recognizing the need for improved training following the Vietnam War, implemented the Red Flag exercises at Nellis AFB. At the heart of this training is the Red Flag Measurement and Debriefing System (RFMDS) and its capability to accurately reconstruct the elements of an intense exercise fought over the deserts of Nevada. This thesis uses the technology of distributed interactive simulation (DIS) to transmit aircraft telemetry onto computer networks, allowing the monitoring and analysis of live Red Flag missions at any site with compatible communications equipment and thesis software. The use of standard DIS protocols enables simulators to "see" Red Flag operations. The three components of the Remote Debriefing Tool bridge reality with simulation. The computer communications component serves as a transparent tap into the RFMDS data stream. The data translation component translates this information into DIS protocol data units (PDUs). PDUs received by the interface and visualization component are used to generate two- and three- dimensional images of the Red Flag environment onto a Silicon Graphics workstation. An extensive set of analysis tools combined with a graphical user interface allows reconstruction of airborne activities, producing more effective and comprehensive training.

14 SUBJECT TERMS DIS, Distributed Interactive Simulation, RFMDS, Red Flag, Mission Analysis, Simulation, Computer Graphics, Debrief	15. NUMBER OF PAGES 217
	16. PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18 SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19 SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- DOD - Leave blank
DOE	- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA	- NASA - Leave blank
NTIS	- NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.